Halbordnungsbasierte Verfeinerung zur Verifikation verteilter Algorithmen

Dissertation

zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (doctor rerum naturalium)
im Fach Informatik

eingereicht an der

Mathematisch-Naturwissenschaftlichen Fakultät II
der Humboldt-Universität zu Berlin

von

Diplom-Mathematikerin

Sibylle Peuker

geboren am 16. Juni 1970 in Dresden

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Jürgen Mlynek

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. sc. nat. Bodo Krause

Gutachter:

1. Prof. Dr. Wolfgang Reisig
2. Prof. Dr. Willem-Paul de Roever
3. Prof. Dr. Peter Starke

eingereicht am: 10. April 2001
Tag der mündlichen Prüfung: 3. Juli 2001

Zusammenfassung

In dieser Arbeit geht es um die schrittweise Verfeinerung verteilter Algorithmen. Dabei wird ein einfacher Algorithmus, der einige gewünschte Eigenschaften hat, Schritt für Schritt zu einem komplexen Algorithmus verfeinert, der konkrete Implementationsanforderungen erfüllt, so daß in jedem Schritt die gewünschten Eigenschaften erhalten bleiben.

Wir stellen einen neuen eigenschaftserhaltenden Verfeinerungsbegriff vor, der auf der kausalen Ordnung der Aktionen eines Algorithmus basiert. Diesen Begriff definieren wir als *Transitionsverfeinerung* für elementare Petrinetze und diskutieren Beweiskriterien. Danach definieren und diskutieren wir die simultane Verfeinerung mehrerer Transitionen.

Zur Modellierung komplexer verteilter Algorithmen sind elementare Petrinetze oft nicht adäquat. Wir benutzen deshalb die in [Rei91] eingeführten algebraischen Petrinetze. Wir definieren Transitionsverfeinerung für algebraische Petrinetze und stellen einen Zusammenhang zur simultanen Verfeinerung von Transitionen in elementaren Petrinetzen her.

Transitionsverfeinerung ist besonders für Verfeinerungsschritte geeignet, in denen synchrone Kommunikation zwischen Agenten durch asynchronen Nachrichtenaustausch ersetzt wird. Wir zeigen dies am Beispiel eines komplexen verteilten Algorithmus, zur Berechnung des minimalen spannenden Baumes in einem gewichteten Graphen [GHS83]. Wir zeigen die Korrektheit dieses Algorithmus in mehreren Schritten, von denen einige Schritte Transitionsverfeinerungen sind. In anderen Schritten sind klassische Verfeinerungsbegriffe ausreichend. Wir übertragen deshalb auch einen klassischen Verfeinerungsbegriff in unser formales Modell.

Abstract

The topic of this PhD thesis is the stepwise refinement of distributed algorithms. Stepwise refinement starts with a simple algorithm with certain desired properties. This algorithm is refined step by step such that the desired properties are preserved in each refinement step. The result is a complex distributed algorithm which satisfies concrete implementation requirements and which still has the desired properties.

We propose a new property preserving notion of refinement which is based on the causal ordering of actions of an algorithm. We call this notion *transition refinement* and we define it first for elementary Petri nets. Furthermore, we discuss proof criteria. Then, we define and discuss the simultaneous refinement of several transitions.

For modelling complex distributed algorithms, we use algebraic Petri nets [Rei91] instead of elementary Petri nets. We define transition refinement for algebraic Petri nets, and we show its relationship to simultaneous transition refinement in elementary Petri nets.

Transition refinement is particularly suitable for refinement steps in which synchronous communication between agents is replaced by asynchronous message passing. We show this by means of a complex distributed algorithm for determining the minimal spanning tree of a weighted graph [GHS83]. We prove the correctness of this

algorithm in several steps. Some of these steps are transition refinements. For other steps, well-known notions of refinement are sufficient. Therefore, we also carry over a well-known notion of refinement into our formal model.

Danksagung

Diese Dissertation ist während meiner Arbeit in den Forschungsprojekten *Verteilte Algorithmen* und *Kompositionale Verifikation von Netzwerkalgorithmen und reaktiven Systemen* entstanden. Beide Forschungsprojekte wurden von der Deutschen Forschungsgemeinschaft gefördert und viele Menschen haben direkt oder indirekt Anteil daran, daß ich diese Arbeit geschrieben habe. Ihnen allen möchte ich an dieser Stelle danken.

Ich danke Prof. Dr. Wolfgang Reisig, der beide Forschungsprojekte geleitet und meine Dissertation betreut hat. Er hat mir die Ästhetik von Petrinetzen und Halbordnungen gezeigt und mich davon überzeugt, daß man alle Wissenschaft auch verständlich darstellen kann und daß es sich lohnt, danach zu streben. Ich habe sehr gerne in der offenen, persönlichen Atmosphäre gearbeitet, die an seinem Lehrstuhl herrscht.

Ich bedanke mich bei allen Mitgliedern der *Kaffeerunde* für alle wissenschaftlichen, persönlichen und allgemeinbildenden Diskussionen, bei allen, die viele kleine Fehler in den Vorversionen meiner Arbeit gefunden haben, besonders bei Ekkart Kindler, der einen großen Fehler gefunden hat und bei Birgit Heene, die für viele Probleme eine Lösung hatte.

Ich danke Prof. Dr. Willem-Paul de Roever und Prof. Dr. Peter Starke für ihre Unterstützung und für die Begutachtung der Arbeit.

Ganz besonders möchte ich mich bei den Menschen bedanken, die mich motiviert haben, die auch meine schlechten Launen ertragen haben und mit mir die schönen Dinge des Lebens geteilt haben; zuallererst bei Hagen, der es immer wieder schafft, meine guten Seiten zum Vorschein zu bringen, bei meinen Eltern, die immer für mich da sind und mich in allem unterstützen, was ich tue, bei meiner ganzen Familie, meinen Freunden, besonders bei Ilka und Dirk für viele schöne Abende und bei Stephan und Juliane, ohne die die letzten Wochen in Berlin eine weniger schöne Erinnerung wären.

Brisbane, im August 2001 Sibylle Peuker

Inhaltsverzeichnis

Einleitung

Die Korrektheit verteilter Algorithmen ist oft schwer nachzuweisen. Lange, von Hand aufgeschriebene Beweise sind fehleranfällig und deshalb oft nicht überzeugend. Die Möglichkeiten maschineller Beweisbarkeit sind begrenzt und insbesondere durch lange Eingaben von Menschen nicht weniger fehleranfällig. Damit ein Beweis überzeugend ist, muß er für einen Menschen (evtl. mit maschineller Hilfe an manchen Stellen) nachvollziehbar sein.

Ein überzeugender Korrektheitsbeweis eines verteilten Algorithmus ist also einerseits formal und andererseits verständlich. Ein formaler Beweis, der verständlich ist, ist gut strukturiert. Eine bekannte Strukturierungsmethode ist schrittweise Verfeinerung. Der Entwurf oder die Verifikation eines verteilten Algorithmus mit schrittweiser Verfeinerung beginnt mit einem einfachen Algorithmus, der einerseits bereits gewisse gewünschte Eigenschaften hat, der aber andererseits noch deklarative, nicht implementierbare Teile enthält. In einer Folge von Verfeinerungsschritten wird dieser Algorithmus so zu einem implementierbaren Algorithmus transformiert, so daß die gewünschten Eigenschaften in jedem Schritt erhalten bleiben.

Es gibt verschiedene Arten von Verfeinerungsschritten, z.B. *Datenverfeinerung*, wobei die Datenrepräsentation eines Algorithmus in eine komplexere Datenrepräsentation transformiert wird, oder *Aktionsverfeinerung*, wobei eine einzelne Aktion durch einen komplexeren Teilalgorithmus ersetzt wird.

In dieser Arbeit zeigen wir, daß es Aktionsverfeinerungsschritte gibt, bei denen es vorteilhaft ist, Halbordnungssemantik zu benutzen, um das Verhalten eines Algorithmus zu beschreiben. Das sind insbesondere Verfeinerungsschritte, in denen man von synchroner zu asynchroner Kommunikation übergeht. Für diese Verfeinerungsschritte schlagen wir einen neuen Verfeinerungsbegriff vor, der auf der kausalen Ordnung von Aktionen in einem Ablauf basiert. Diesen Begriff nennen wir *Transitionsverfeinerung*.

Wir demonstrieren anhand eines komplexen verteilten Algorithmus, des Algorithmus von Gallager, Humblet und Spira (kurz: GHS-Algorithmus, [GHS83]), wie Transitionsverfeinerung vorteilhaft eingesetzt wird. Zur schrittweisen Verfeinerung des Algorithmus kombinieren wir Transitionsverfeinerung mit einem klassischen Verfeinerungsbegriff. Dazu übertragen wir einen klassischen Verfeinerungsbegriff, der auf sequentieller Semantik basiert, in unser formales Modell.

Verteilte Algorithmen

Ein Algorithmus ist *verteilt*, wenn er für eine *verteilte Architektur* formuliert ist. Eine verteilte Architektur besteht aus mehreren Komponenten, die wir *Agenten* nennen.

Ein verteilter Algorithmus beschreibt das Verhalten der Agenten zur gemeinsamen Lösung einer Aufgabe oder zur Erreichung individueller Ziele einzelner Agenten, z.B. bei der Verteilung knapper Ressourcen.

Wir nennen eine verteilte Architektur ein *Kommunikationsnetzwerk*, wenn seine Agenten keinen gemeinsamen Speicher haben, sondern ausschließlich über Kommunikationskanäle miteinander verbunden sind. Verteilte Algorithmen für solche Architekturen heißen *nachrichtenbasiert*. Wir konzentrieren uns in dieser Arbeit auf *nachrichtenbasierte* Algorithmen. Aus vielen Monographien über verteilte Algorithmen geht hervor, daß nachrichtenbasierte Algorithmen eine zentrale Rolle unter den verteilten Algorithmen spielen [AW98, Bar96, Lyn96, Mat89, Ray88, Rei98, Tel94, Tel91].

Speziell zu nachrichtenbasierten Algorithmen bemerken Lamport und Lynch in [LL90]: "In general, network algorithms are typically longer and harder to understand than other types of distributed algorithms we are considering, and rigorous correctness proofs are seldom given." In dieser Arbeit geht es nicht darum eine umfangreiche, vollständige Theorie auf der Grundlage eines neuen Verfeinerungsbegriffs zu bilden. Es geht darum, einen Begriff zu formulieren, mit dem man Phänomene in komplexen verteilten Algorithmen beschreiben kann, um solche Algorithmen besser zu verstehen und deren Korrektheit nachzuweisen.

Schrittweise Verfeinerung

Schrittweise Verfeinerung wurde erstmals von Wirth in [Wir71] als Technik zur Programmentwicklung beschrieben. Wirth schreibt dort: "... the program is gradually developed in a sequence of *refinement steps*. In each step, one or several instructions of the given program are decomposed into more detailed instructions." Inzwischen besteht Konsens darüber, daß eigenschaftserhaltende Verfeinerungsbegriffe unerläßlich für Entwurf, Verständnis und Verifikation verteilter Algorithmen sind.

Es gibt verschiedene Kriterien dafür, wann ein Verfeinerungsschritt korrekt ist. Das gebräuchlichste Kriterium ist, daß jeder Ablauf des verfeinerten Systems auch ein Ablauf des Ausgangssystems ist. Da Eigenschaften von verteilten Algorithmen meist so spezifiziert werden, daß sie in jedem Ablauf gelten, ist dieser Verfeinerungsbegriff sehr sinnvoll: Alle Eigenschaften des Ausgangssystems bleiben bei einer Verfeinerung erhalten. Es wurden verschiedene Beweismethoden für diesen Verfeinerungsbegriff vorgeschlagen, z.B. Vorwärtssimulation oder *history variables* [Jon91, AL91, LT87], Rückwärtssimulation oder *prophecy variables* [Jon91, HHS87, AL91].

In vielen Fällen ist aber der oben beschriebene Verfeinerungsbegriff zu restriktiv oder nicht adäquat. In [GKS92] wird ein weniger restriktiver Verfeinerungsbegriff definiert. Das dort beschriebene *interface refinement* erlaubt sehr komplexe Verfeinerungsschritte. Der *refinement calculus* [Bac88, ME92] ist ein logischer Ansatz, der eine Art Berechnungskalkül angibt, mit dem man aus einer korrekten Spezifikation eine verfeinerte korrekte Spezifikation berechnen kann. Superposition [CM88, BS96] beschreibt das Aufsetzen eines Programms auf ein anderes Programm. In [Sie96] wird *delayed simulation* als passende Beweismethode für selbststabilisierende Systeme beschrieben. *Atomicity refinement* [Gri93, Gri96] ist ein Verfeinerungsbegriff, bei

dem ein System so verfeinert wird, daß wichtige Invarianten des Systems erhalten bleiben.

Halbordnungsbasierte Verfeinerungsbegriffe

Oft wird das Verhalten eines verteilten Algorithmus durch sequentielle Abläufe beschrieben, d.h. durch Sequenzen von Zuständen und Aktionen. Durch diese Beschreibung geht Information über das Verhalten des Algorithmus verloren. Die Beschreibung des Verhaltens durch Halbordnungen enthält mehr Information. In einer Halbordnungssemantik werden die kausalen Zusammenhänge zwischen Aktionen des Algorithmus beschrieben. Dadurch können wir insbesondere *Nebenläufigkeit* von Aktionen beschreiben.

Eigenschaftserhaltende Verfeinerungsbegriffe, die auf Halbordnungssemantik basieren, findet man insbesondere für Petrinetze, da Petrinetze eine kanonische Halbordnungssemantik haben. Solche Begriffe beziehen sich meist auf die Verfeinerung einer einzelnen Stelle oder einer einzelnen Transition des Petrinetzes [BGV90]. In [Vog87] definiert Vogler ein *Modul* als Verfeinerung einer Transition in jeder beliebigen Umgebung. Damit entwickelt Vogler eine Idee von Valette [Val79] weiter. In [KP99] wird beschrieben, wie ein Algorithmus mit einfachen syntaktischen Regeln in eine Umgebung eingebettet werden kann, so daß alle Eigenschaften des Algorithmus erhalten bleiben. Diese Verfeinerungsbegriffe für Petrinetze haben die gleiche Restriktion: Es ist nicht möglich, durch die Verfeinerung mehr Nebenläufigkeit in das System einzuführen, d.h. z.B. synchrone durch asynchrone Kommunikation zu ersetzen.

Viele andere halbordnungsbasierte Verfeinerungsbegriffe sind nicht zur schrittweisen Verfeinerung geeignet, weil die Aktionen uninterpretiert sind und die Korrektheit eines Verfeinerungsschrittes nicht definierbar ist [BT97, vGG90, vGG89, Vog93].

Transitionsverfeinerung

Die in dieser Arbeit definierte Transitionsverfeinerung ist passend für Verfeinerungsschritte, in denen zusätzliche Nebenläufigkeit in ein System eingeführt wird. Das ist der Fall, wenn eine synchrone Aktion mehrerer Agenten durch asynchronen Nachrichtenaustausch zwischen den Agenten ersetzt wird.

Betrachten wir beispielsweise einen Algorithmus A, der eine atomare Aktion enthält, bei der in einem Schritt alle Agenten eines Netzwerks eine Nachricht erhalten. Diese Aktion ist auf einer verteilten Architektur nicht implementierbar. Sie soll deshalb durch einen verteilten Broadcast-Algorithmus ersetzt werden. Wir nehmen an, daß der Algorithmus A nicht-deterministisch ist und insbesondere nicht festlegt, *wie oft* eine Benachrichtigung aller Agenten erforderlich ist. Die vorher atomare Aktion der Benachrichtigung wird also durch mehrere teilweise nebenläufig zueinander ausgeführte Aktionen ersetzt. Vorher konnte die Benachrichtigung nur ausgeführt werden, wenn alle Agenten dazu bereit waren. Nach der Ersetzung kann es passieren, daß einige Agenten bereits mit dem Broadcast-Algorithmus beginnen, andere Agenten aber nicht dazu bereit sind und der Algorithmus deshalb unvollendet abgebrochen wird. Es kann also passieren, daß durch die zusätzliche Nebenläufigkeit neue Konflikte mit der Umgebung eingeführt werden.

Sei A' der Algorithmus, der aus der Ersetzung der Benachrichtigungsaktion durch den Broadcast-Algorithmus in A resultiert. Ob A' korrekt arbeitet, hängt davon ab, wie der Broadcast-Algorithmus mit seiner Umgebung verwoben ist. Die Eigenschaften von A' sind kompositional aus den Eigenschaften von A und den Eigenschaften des Broadcast-Algorithmus ableitbar, wenn jeder begonnene Broadcast ohne Interaktion mit der Umgebung beendet wird.

Dies kann man adäquat mit verteilten Abläufen beschreiben: In jedem verteilten Ablauf gibt es zu jedem Broadcast zwei Schnitte[1], zwischen denen genau die zu diesem Broadcast gehörenden Aktionen liegen.

Die Benutzung von Schnitten zur Kennzeichnung verschiedener Phasen in einem verteilten Ablauf, wird auch in [vdMM98] vorgeschlagen. Diese Sichtweise auf ein verteiltes System ist mit den von Elrad und Francez in [EF82] eingeführten *Communication Closed Layers* und darauf aufbauenden Arbeiten [Zwi98, SdR94, JMZ91, CG88] verwandt. Wir diskutieren dies genauer im letzten Abschnitt dieser Arbeit. Logisch kann man eine ähnliche Sicht auf ein System auch mit *Interleaving Set Temporal Logic* von Katz und Peled [KP90, KP92] beschreiben.

Der GHS-Algorithmus

Der Algorithmus von Gallager, Humblet und Spira ist ein verteilter Algorithmus zur Berechnung des minimalen spannenden Baumes in einem gewichteten Graphen. Die Korrektheit des Algorithmus wurde schon mehrfach nachgewiesen, aber die Beweise sind meist schwer nachvollziehbar oder unvollständig.

Wegen seiner Komplexität hat sich dieser Algorithmus zu einem Prüfstein für Verifikationsmethoden verteilter Algorithmen entwickelt. Immer wieder wurden neue Beweismethoden [CG88, Hes99a, SdR87, WLL88] an diesem Algorithmus ausprobiert. Auch wenn nicht jeder Beweis vollständig zufriedenstellend ist, wurde durch die Arbeit daran jedes Mal eine neue Denkweise hervorgebracht.

Die mathematische Idee, die dem Algorithmus zugrunde liegt, ist einfach. Deshalb gibt es eine leicht verständliche, abstrakte Sichtweise. Die Schwierigkeit, den GHS-Algorithmus zu verstehen, erklärt sich vor allem aus der komplexen Kommunikation zwischen den Agenten und dem hohen Grad an Nebenläufigkeit.

Der Algorithmus ist bei einer monolithischen Betrachtungsweise schwer verständlich. Wir geben eine schrittweise verfeinernde Herleitung des Algorithmus an, die die Komplexität des Algorithmus in leicht verständliche Schritte auflöst. Insbesondere werden durch die Benutzung von Halbordnungssemantik Eigenschaften des Algorithmus offensichtlich, die zum Verständnis beitragen und die bei ausschließlicher Benutzung sequentieller Semantik schwer zu erkennen sind. In dieser Herleitung sind vier der zwölf Verfeinerungsschritte Transitionsverfeinerungen, deren Korrektheit mit klassischen Verfeinerungsmethoden nicht beschreibbar ist. In diesen vier Transitionsverfeinerungsschritten wird immer eine synchrone Aktion mehrerer Agenten durch asynchronen Nachrichtenaustausch ersetzt.

[1] Ein Schnitt ist ein verteilter globaler Zustand.

Aufbau der Arbeit

Die Arbeit besteht aus drei Teilen:

Im ersten Teil wird Transitionsverfeinerung als neuer halbordnungsbasierter Verfeinerungsbegriff eingeführt und diskutiert. Um die Idee und die wichtigsten Charakteristika verständlich zu machen, definieren wir den Begriff zuerst für die einfachste Version von Petrinetzen. Wir diskutieren Kriterien dafür, wann die Ersetzung einer einzelnen Transition durch ein Petrinetz eine korrekte Transitionsverfeinerung ist. Dann definieren und diskutieren wir die simultane Verfeinerung mehrerer Transitionen. Diese simultane Transitionsverfeinerung ist die Grundlage für die Verfeinerung von Transitionen in algebraischen Petrinetzen.

Im zweiten Teil der Arbeit diskutieren wir Modellierung und Verifikation von verteilten Algorithmen, die auf asynchronem Nachrichtenaustausch basieren. Zur Modellierung komplexer verteilter Algorithmen sind einfache Petrinetze oft nicht adäquat. Wir benutzen deshalb algebraische Petrinetze [Rei91], die eine kompakte Darstellung komplexer verteilter Algorithmen erlauben. Wir führen im zweiten Teil der Arbeit Transitionsverfeinerung für algebraische Petrinetze ein. Außerdem übertragen wir den klassischen Verfeinerungsbegriff von Abadi und Lamport [AL91] auf algebraische Petrinetze.

Im dritten Teil der Arbeit verwenden wir die im zweiten Teil vorgestellten Verfeinerungsbegriffe und Ergebnisse zur Herleitung des GHS-Algorithmus.

Teil I

Verfeinern mit Halbordnungen

In diesem Teil der Arbeit führen wir einen neuen Verfeinerungsbegriff ein, der auf Halbordnungen basiert: *Transitionsverfeinerung*. Um die Idee und die wichtigsten Charakteristika verständlich zu machen, definieren wir den Begriff zuerst für die einfachste Version von Petrinetzen.

Wir diskutieren Kriterien dafür, wann die Ersetzung einer einzelnen Transition durch ein Petrinetz eine Transitionsverfeinerung ist. Dann definieren und diskutieren wir die simultane Verfeinerung mehrerer Transitionen. Diese simultane Transitionsverfeinerung ist die Grundlage für die Verfeinerung von Transitionen in algebraischen Petrinetzen.

1 Elementare Transitionsverfeinerung

1.1 Telefongespräch versus E-Mail

Wir erläutern in diesem Abschnitt den Begriff *Transitionsverfeinerung* informell anhand eines Beispiels. In diesem Beispiel benutzen wir Petrinetze zur Illustration. Formal führen wir Petrinetze im nächsten Abschnitt ein.

Transitionsverfeinerung kann besonders dann sinnvoll eingesetzt werden, wenn eine synchrone Kommunikation zwischen Agenten durch eine asynchrone Kommunikation ersetzt wird. Eine typische synchrone Kommunikation zwischen zwei Agenten ist ein Telefongespräch. Beide Agenten führen diese Kommunikation gemeinsam und gleichzeitig aus. Eine typische asynchrone Kommunikation ist das Versenden und Empfangen einer E-Mail oder eines Briefes. Die Nachricht wird von einem Agenten versendet und von dem anderen Agenten irgendwann später empfangen.

Wir betrachten nun das Telefongespräch als Teil eines größeren Systems. Ob und wie ein Telefongespräch zwischen zwei Agenten durch den Austausch von E-Mails ersetzt werden kann, hängt davon ab, in welchem kausalen Zusammenhang die anderen Aktionen der Agenten im System zu dem Telefongespräch stehen.

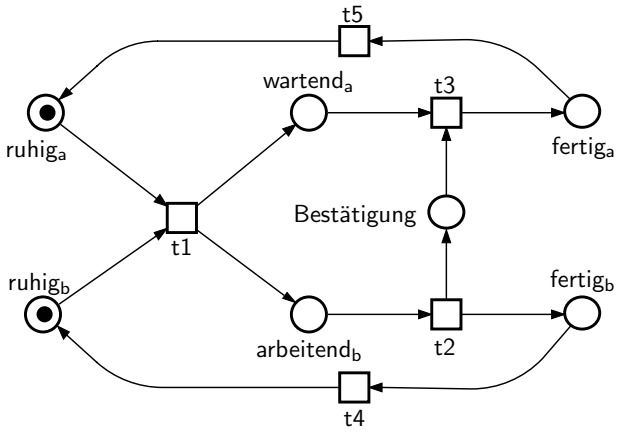


Abb. 1.1: Das Protokoll Σ_1

In Abb. 1.1 ist ein Protokoll zwischen zwei Agenten a und b dargestellt. Im Anfangszustand sind beide Agenten ruhig. Beide Agenten können dann miteinander telefonieren (Transition $t1$). Wir können uns dabei vorstellen, daß der Agent a dem Agenten b einen Auftrag gibt. Nach dem Telefongespräch wartet a und b arbeitet. Wenn b den Auftrag ausgeführt hat, schickt er eine Bestätigung an a und ist fertig (Transition $t2$). Wenn a die Bestätigung erhält, daß der Auftrag ausgeführt wurde,

wird er ebenfalls fertig (Transition $t3$). Ein Agent der fertig ist, wird wieder ruhig (Transitionen $t4$ und $t5$).

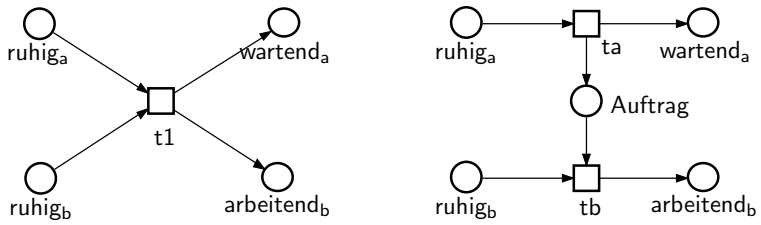


Abb. 1.2: Die Transition $t1$ und das Ersetzungsnetz N

Das Telefongespräch soll nun dadurch ersetzt werden, daß a dem Agenten b einen Auftrag per E-Mail erteilt. In Abb. 1.2 haben wir ein Telefongespräch isoliert dargestellt und daneben das aufeinanderfolgende Versenden und Empfangen eines Auftrags durch das Petrinetz N modelliert.

In Isolation verhält sich das Petrinetz N wie die Transition, womit wir Folgendes meinen: Wenn in N der Vorbereich von $t1$ mit je einer Marke markiert ist (also eine Marke auf $\mathsf{ruhig_a}$ und eine auf $\mathsf{ruhig_b}$ liegt) und alle anderen Stellen unmarkiert sind, dann ist jeder Ablauf von N endlich und im Endzustand ist gerade der Nachbereich von $t1$ mit je einer Marke markiert (eine Marke auf $\mathsf{wartend_a}$ und eine auf $\mathsf{arbeitend_b}$). Wegen dieser Eigenschaft nennen wir N ein *Ersetzungsnetz* für $t1$.

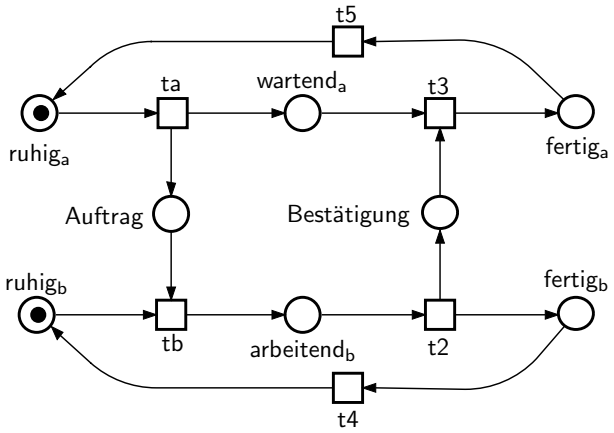


Abb. 1.3: Das Protokoll Σ_1'

Das System Σ_1' (Abb. 1.3) erhalten wir, indem wir im System Σ_1 die Transition $t1$ durch das Ersetzungsnetz N ersetzen. Ob N auch eine (eigenschaftserhaltende) Verfeinerung ist, hängt von dem System ab, in dem wir $t1$ durch N ersetzt haben. Im System Σ_1 ist N tatsächlich eine Transitionsverfeinerung, denn die Systeme Σ_1 und Σ_1' verhalten sich ähnlich. In beiden Systemen werden z.B. beide Agenten immer wieder ruhig. Wir erklären später noch genauer, was "ähnliches Verhalten" bedeutet.

Wir zeigen nun, daß Transitionsverfeinerung von der Umgebung abhängig ist, indem wir das System nur wenig verändern. Das System Σ_2 arbeitet so wie Σ_1, mit dem Unterschied, daß der Agent b nach Beendigung eines Auftrags aus dem System ausscheiden kann (Transition $t6$).

Wenn b ausscheidet, kann a keinen Auftrag mehr erteilen, da kein Telefongespräch mehr stattfinden kann. Nach dem Ausscheiden von b bleibt a für immer im Zustand ruhig.

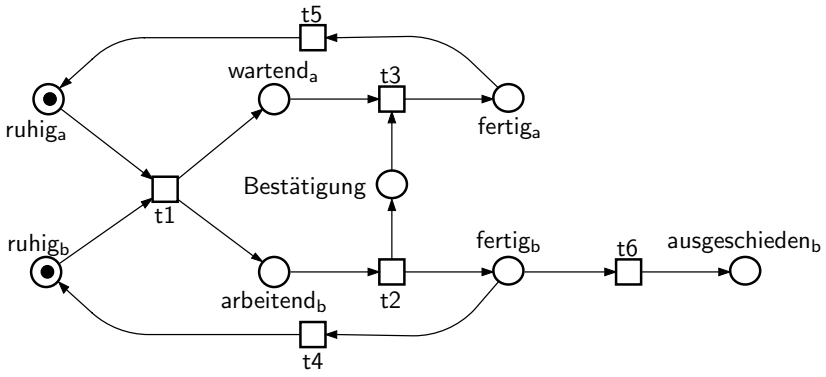


Abb. 1.4: Das Protokoll Σ_2

Wenn wir in diesem System das Telefongespräch durch eine E-Mail ersetzen, erhalten wir das System Σ_2' in Abb. 1.5.

Wenn b in diesem System ausscheidet, kann a immer noch eine E-Mail mit einem Auftrag abschicken, wartet dann aber für immer, denn der Auftrag wird nie bestätigt. Das ist ein Unterschied zum Verhalten von Σ_2. Die Systeme Σ_2 und Σ_2' verhalten sich in unserem (noch immer nicht erklärten) Sinne nicht ähnlich. In Σ_2 ist eine E-Mail also keine eigenschaftserhaltende Verfeinerung für ein Telefongespräch.

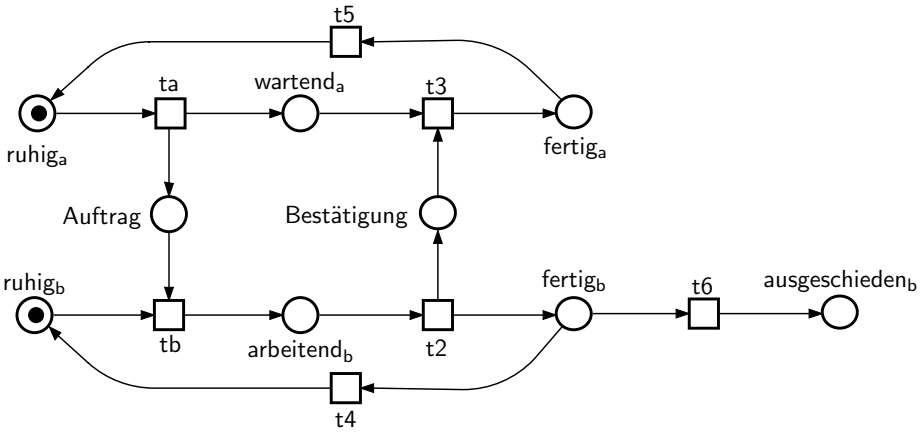


Abb. 1.5: Das Protokoll Σ_2'

Man könnte auch einen Verfeinerungsbegriff definieren, in dem Σ_2' eine Verfeinerung von Σ_2 ist. Wir sind an einem strengeren Verfeinerungsbegriff interessiert, der Σ_2' nicht als Verfeinerung von Σ_2 zuläßt.

Bevor wir das Verhalten der Systeme genauer analysieren, geben wir noch ein Beispiel an, in dem deutlich wird, daß aus derselben Ersetzung noch wesentlich "unähnlicheres" Verhalten resultieren kann.

Im System Σ_3 werden die Aufträge nicht mehr bestätigt. Ansonsten arbeitet das

System so wie Σ_2. Agent a ist bereits nach dem Telefongespräch fertig und Agent b schickt keine Bestätigung wenn er fertig wird (Abb. 1.6).

Ähnlich wie in Σ_2 wird a noch ein letztes Mal ruhig, wenn b ausscheidet und kann dann auch nicht mehr schalten. Der Agent a bleibt dann auch für immer ruhig.

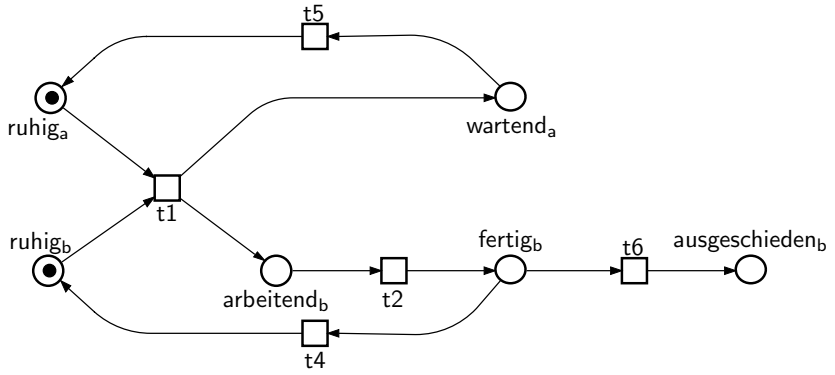


Abb. 1.6: Das Protokoll Σ_3

Wenn wir in diesem System das Telefongespräch durch eine E-Mail ersetzen, erhalten wir das System Σ_3' in Abb. 1.7. In jedem Ablauf des Systems kann a immer weiter Aufträge erteilen, auch wenn b schon längst ausgeschieden ist. Die Systeme Σ_3 und Σ_3' verhalten sich überhaupt nicht ähnlich. Auch hier ist eine E-Mail keine eigenschaftserhaltende Verfeinerung für ein Telefongespräch.

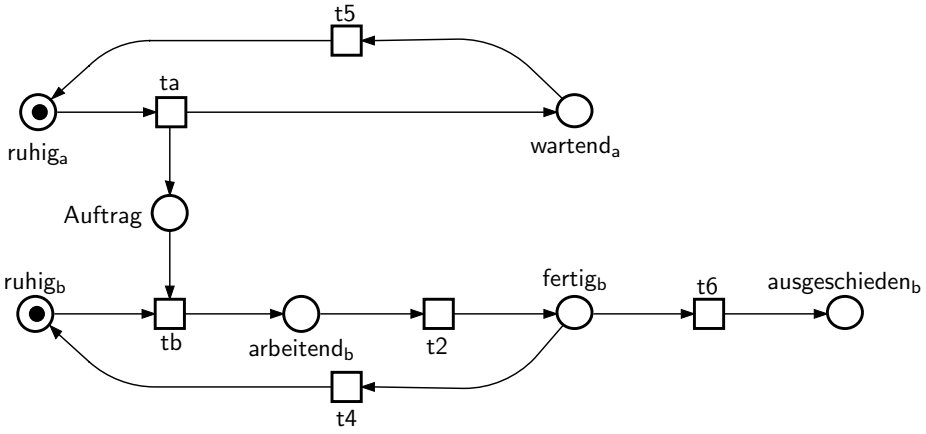


Abb. 1.7: Das Protokoll Σ_3'

Wir erklären nun genauer, was "ähnliches Verhalten" bedeutet. Wir beschreiben das Verhalten von Systemen durch *verteilte Abläufe*, die wir formal im nächsten Abschnitt erklären. In Abb. 1.8 ist ein Präfix eines unendlichen verteilten Ablaufs von Σ_1 dargestellt.

Einen verteilten Ablauf eines Systems modellieren wir mit Hilfe eines *Kausalnetzes*. Ein Kausalnetz ist ein spezielles, azyklisches Petrinetz mit nicht-verzweigenden Stellen, d.h. Stellen, die höchstens eine eingehende und eine ausgehende Kante haben. Die Stellen eines Kausalnetzes nennen wir *Bedingungen* und die Transitionen nennen

wir *Ereignisse*. Wir beschriften jedes Ereignis mit dem Namen einer Transition des Systems und jede Bedingung mit dem Namen einer Stelle. In Abb. 1.8 haben wir die Beschriftungen durch ihre Anfangsbuchstaben abgekürzt. Durch diese Beschriftung können wir jedes Ereignis als *Auftreten* einer Transition von Σ_1 erkennen.

Unser Beispielablauf in Abb. 1.8 beginnt im Anfangszustand des Systems: beide Agenten sind ruhig. Dann telefonieren die beiden Agenten und a wartet und b arbeitet. Agent b wird fertig und schickt eine Bestätigung. Agent a wird fertig und beide Agenten werden wieder ruhig. Danach telefonieren wieder beide und so weiter... (angedeutet, durch die drei Punkte auf der rechten Seite).

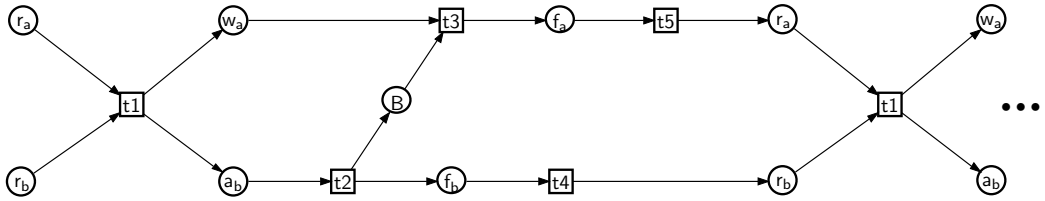


Abb. 1.8: Der verteilte Ablauf ρ_1 von Σ_1

Ein verteilter Ablauf ist maximal bzgl. des Schaltens von Transitionen, d.h. er ist nicht beendet, solange noch eine Transition aktiviert ist. In einem verteilten Ablauf sind alle Konflikte[1] aufgelöst. In Σ_1 gibt es keine Konflikte, deshalb gibt es auch nur einen einzigen verteilten Ablauf. Das erste Auftreten der Transition $t4$ und das erste Auftreten der Transition $t5$ sind im obigen Ablaufstück nicht kausal geordnet. Wir sagen die beiden Ereignisse treten *nebenläufig zueinander* auf. Dasselbe gilt für das erste Auftreten von $t3$ und $t4$.

In einem sequentiellen Ablauf sind alle Ereignisse total geordnet. Das System Σ_1 hat unendlich viele sequentielle Abläufe, je nachdem wie die Transitionen $t4$ und $t5$ in jeder Runde geordnet werden. Dies ist ein Beispiel dafür, daß man an verteilten Abläufen mehr Eigenschaften eines Systems erkennen kann, als an sequentiellen: In einem System, das genau einen verteilten Ablauf hat, treten keine Konflikte auf. Aus den sequentiellen Abläufen des Systems kann man nicht auf die Konfliktfreiheit schließen.

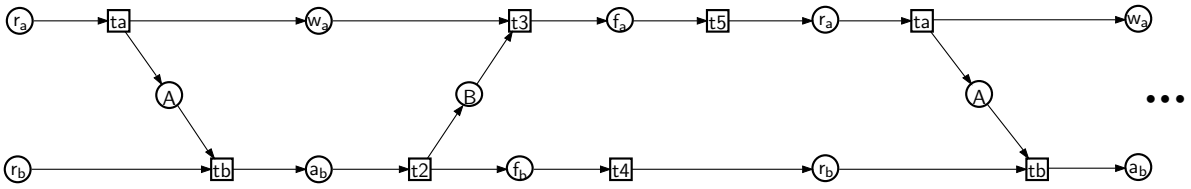


Abb. 1.9: Der verteilte Ablauf ρ_1' von Σ_1'

Wir erhalten den Ablauf ρ_1' von Σ_1', indem wir im Ablauf ρ_1 von Σ_1 jedes Auftreten von $t1$ durch den Ablauf des Ersetzungsnetzes N aus Abb. 1.2 ersetzen.

[1] Ein Konflikt zwischen zwei Transitionen besteht, wenn jede der beiden Transitionen schalten kann und wenn es eine Stelle gibt, die ausgehende Kanten zu beiden Transitionen hat (siehe Abschnitt 1.2).

Allgemeiner sagen wir, daß sich Σ_1 und Σ_1' ähnlich verhalten, wenn jeder verteilte Ablauf von Σ_1' aus einem verteilten Ablauf von Σ_1 konstruiert werden kann, indem man jedes Ereignis, das mit t1 beschriftet ist, durch einen verteilten Ablauf von N ersetzt (siehe Abb. 1.9), und wenn umgekehrt, jeder Prozeß, der so konstruiert wurde, ein verteilter Ablauf von Σ_1' ist. Wenn das der Fall ist, dann nennen wir das Ersetzungsnetz N eine *Transitionsverfeinerung* von t1 im System Σ_1. In Σ_1 ist das der Fall, denn Σ_1' hat auch nur den einen verteilten Ablauf (Abb. 1.9), den man nach obiger Anleitung erhält.

Eigenschaften verteilter Systeme wie "in diesem System gilt immer..." oder "in diesem System gilt irgendwann..." sind über die Abläufe des Systems definiert. Wenn zwei Systeme ähnliche verteilte Abläufe haben, dann haben sie auch ähnliche Eigenschaften. In den beiden Systemen Σ_1 und Σ_1' werden z.B. beide Agenten immer wieder ruhig, und beide Systeme terminieren nicht.

Betrachten wir nun noch einmal das zweite Beispiel. Einen verteilten Ablauf von Σ_2', in dem b ausscheidet, kann man nicht aus einem verteilten Ablauf von Σ_2 nach der obigen Anleitung erhalten, denn wenn a den letzten Auftrag abschickt, wird dieser nie bestätigt. Das ist, als hätte die Transition $t1$ nur "halb" geschaltet. Für dieses letzte Schalten von ta gibt es also keine Entsprechung in einem Ablauf von Σ_2. Aus diesem Grund ist N auch keine Transitionsverfeinerung von $t1$ in Σ_2.

In Σ_3 ist der Unterschied noch deutlicher: Jeder Ablauf von Σ_3, in dem b ausscheidet, ist endlich. In Σ_3' dagegen ist jeder Ablauf unendlich, unabhängig davon, ob b ausscheidet oder nicht. Wenn b ausscheidet, schickt a unendlich oft neue Aufträge. Hier ist es so, als schaltete die Transition $t1$ unendlich oft nur "halb". In Σ_3 ist N also auch keine Transitionsverfeinerung von $t1$. Wenn man synchrone Kommunikation durch asynchrone Kommunikation verfeinern will, kann man dies nicht unabhängig von der Umgebung tun. Unter gewissen Bedingungen ist es jedoch möglich, synchrone durch asynchrone Kommunikation zu verfeinern, so daß wichtige Eigenschaften des Systems erhalten bleiben. Durch Transitionsverfeinerung kann man adäquat ausdrücken, wann dies der Fall ist.

1.2 Petrinetze und verteilte Abläufe

In diesem Abschnitt definieren wir die im letzten Abschnitt bereits informell benutzten Begriffe. Wir definieren elementare Petrinetze (S/T-Netze ohne Kantengewichtung) und deren Halbordnungssemantik. Der Leser, der mit dieser Petrinetzklasse und verteilten Abläufen (z.B. aus [Rei85] oder [Rei98]) vertraut ist, kann diesen Abschnitt überspringen.

Definition 1.1 (Petrinetz)

Ein *Petrinetz* (kurz: *Netz*) $N = (P, T, F)$ besteht aus zwei nicht-leeren, disjunkten Mengen P und T und der Flußrelation $F \subseteq (T \times P) \cup (P \times T)$, so daß für jedes $t \in T$ die Mengen ${}^\bullet t = \{p \in P \mid (p,t) \in F\}$ und $t^\bullet = \{p \in P \mid (t,p) \in F\}$ nicht leer und endlich sind. Die Elemente von P, T und F nennen wir Stellen, Transitionen bzw. Kanten des Netzes. ⋆

Für jedes $x \in P \cup T$ nennen wir die Menge ${}^\bullet x = \{y \in P \cup T \mid (y,x) \in F\}$ den *Vorbereich* von x und die Menge $x^\bullet = \{y \in P \cup T \mid (x,y) \in F\}$ den *Nachbereich*

von x. Zur Unterscheidung von algebraischen Petrinetzen nennen wir ein wie oben definiertes Petrinetz manchmal auch *elementares Petrinetz*.

Für eine technisch angenehme Präsentation unserer Ergebnisse halten wir in unserer Arbeit eine universelle Stellenmenge $\mathcal{P}$ fest und nehmen an, daß alle in dieser Arbeit betrachteten Petrinetze nur Stellen aus dieser Menge haben, d.h. $P \subseteq \mathcal{P}$.

Definition 1.2 (Markierung)

Eine *Markierung* ist eine Funktion $M : \mathcal{P} \longrightarrow \mathbb{N}$, die höchstens endlich vielen Stellen eine von Null verschiedene natürliche Zahl zuordnet. Eine *Markierung des Netzes* N ist eine Markierung, für die $M(p) = 0$ für alle $p \in \mathcal{P} \setminus P$ gilt. ⋆

Eine Markierung eines Petrinetzes beschreibt einen globalen Zustand des Petrinetzes. Wir visualisieren eine Markierung durch Marken; jede Stelle trägt die durch die Markierung angegebene Anzahl von Marken. Obwohl wir Netze mit unendlich vielen Stellen zulassen, lassen wir nach der obigen Definition nur endliche Markierungen zu, d.h. in jeder Markierung liegen nur auf endlich vielen Stellen Marken. Jeder Transition t ordnen wir die beiden Markierungen t^- und t^+ zu, die folgendermaßen definiert sind:

$$t^-(p) = \begin{cases} 1 & \text{für } p \in {}^\bullet t \\ 0 & \text{sonst} \end{cases} \qquad t^+(p) = \begin{cases} 1 & \text{für } p \in t^\bullet \\ 0 & \text{sonst} \end{cases}$$

Wir definieren die *Addition von Markierungen* punktweise, d.h. für zwei Markierungen M_1 und M_2 des Netzes N, definieren wir die Summe $M_1 + M_2 : \mathcal{P} \longrightarrow \mathbb{N}$ durch $(M_1 + M_2)(p) = M_1(p) + M_2(p)$ für alle $p \in \mathcal{P}$. Analog vergleichen wir Markierungen punktweise und schreiben $M_1 \leq M_2$, falls $M_1(p) \leq M_2(p)$ für alle $p \in \mathcal{P}$.

Eine Markierung wird durch das *Schalten* einer Transition verändert. Eine Transition t ist *aktiviert* in der Markierung M, wenn auf jeder Stelle im Vorbereich von t mindestens eine Marke liegt, d.h. $t^- \leq M$. Eine aktivierte Transition kann schalten. Das führt zur Markierung M', die durch $M' + t^- = M + t^+$ gegeben ist. Den gerade beschriebenen Vorgang nennen wir *Schritt* und bezeichnen ihn durch $M \xrightarrow{t} M'$. Eine Markierung M' ist von der Markierung M aus *erreichbar*, wenn es eine endliche, ggf. leere Folge von Schritten $M_0 \xrightarrow{t_1} M_1 \ldots \xrightarrow{t_n} M_n$ mit $M = M_0$ und $M' = M_n$ gibt. Zwei verschiedene Transitionen t_1 und t_2 stehen in *Konflikt* in M, wenn beide in M aktiviert sind, aber ihre Vorbereiche nicht disjunkt sind.

Ein Paar $\Sigma = (N, M_0)$ aus einem Petrinetz N und einer Markierung M_0 von N nennen wir ein *elementares System*. Die Markierung M_0 beschreibt den Anfangszustand des Systems, und wir nennen M_0 die Anfangsmarkierung von Σ. Eine Markierung M ist *erreichbar* in Σ, wenn M von M_0 aus erreichbar ist. Wir verwenden den Systembegriff auch für die später zu definierenden algebraischen Petrinetze. Wenn eine Unterscheidung nicht wichtig ist oder aus dem Kontext hervorgeht, schreiben wir aber nur *System*, statt *elementares System* oder *algebraisches System*.

Definition 1.3 (Sicheres System)

Ein System ist *sicher*, wenn auf keiner Stelle jemals mehr als eine Marke liegt, d.h. für jede erreichbare Markierung M und für jede Stelle $p \in \mathcal{P}$ gilt $M(p) \leq 1$. ⋆

Wir definieren nun eine Halbordnungssemantik für Petrinetze mit Hilfe verteilter Abläufe. Für alle Systeme nehmen wir an, daß jede aktivierte Transition irgendwann

schaltet oder eine zu der Transition in Konflikt stehende Transition schaltet. Diese *Progressannahme* formalisieren wir in der Definition verteilter Abläufe. Ein verteilter Ablauf wird durch ein Kausalnetz dargestellt. Ein Kausalnetz ist ein Petrinetz mit besonderen Eigenschaften. Es ist azyklisch, d.h. die Flußrelation enthält keinen Kreis (vgl. Anhang A.2). Außerdem ist ein Kausalnetz an Stellen unverzweigt, d.h. jede Stelle hat höchstens eine eingehende und eine ausgehende Kante. Die Stellen eines Kausalnetzes nennen wir *Bedingungen* und die Transitionen nennen wir *Ereignisse*.

Definition 1.4 (Kausalnetz)

Ein Netz $K = (B, E, \lessdot)$ ist ein *Kausalnetz*, wenn folgende Bedingungen gelten:

(i) K ist stellenunverzweigt, d.h. $|{}^\bullet b| \leq 1$ und $|b^\bullet| \leq 1$ für jede Bedingung $b \in B$.

(ii) Die Menge ${}^\circ\mathrm{K} = \{b \in B \mid {}^\bullet b = \emptyset\}$ ist nicht leer und endlich.

(iii) Die Relation $\lessdot$ ist azyklisch. Dadurch ist die reflexive transitive Hülle von $\lessdot$ eine Halbordnung, die wir mit $\leq$ bezeichnen.

(iv) Jedes Element von $B \cup E$ hat nur endlich viele Vorgänger bzgl. der Halbordnung $\leq$. ⋆

Wir definieren nun einige Begriffe, um Aussagen über Kausalnetze zu treffen. Ein *Schnitt* ist eine maximale, aber endliche Menge von paarweise unabhängigen Bedingungen bzgl. der Kausalordnung $\leq$ eines Kausalnetzes. Zwei voneinander unabhängige Ereignisse bzgl. $\leq$ nennen wir auch zueinander *nebenläufig*.

Definition 1.5 (co-Menge, Schnitt, nebenläufige Ereignisse)

Eine Menge $C \subseteq B$ ist eine *co-Menge*[2] des Kausalnetzes $K = (B, E, \lessdot)$ wenn für je zwei Bedingungen $b_1, b_2 \in C$ gilt: $b_1 \not\lessdot b_2$ oder $b_1 = b_2$. Eine maximale, endliche co-Menge C ist ein *Schnitt*.

Zwei verschiedene Ereignisse $e_1, e_2 \in E$ sind zueinander *nebenläufig*, wenn $e_1 \not\leq e_2$ und $e_1 \not\geq e_2$. Wir bezeichnen dies mit e_1 co e_2. ⋆

(Bemerkung: In der Literatur werden co-Mengen meist für Bedingungen und Ereignisse definiert. Wir werden jedoch nur co-Mengen von Bedingungen benutzen.)

Die Menge ${}^\circ\mathrm{K} = \{b \in B \mid {}^\bullet b = \emptyset\}$ ist ein Schnitt. Diesen Schnitt nennen wir den *Anfangsschnitt* von K. Die Menge $\mathrm{K}^\circ = \{b \in B \mid b^\bullet = \emptyset\}$ ist eine co-Menge. Wir nennen K° das *Ende* des Kausalnetzes. Das Ende eines Kausalnetzes K ist genau dann ein Schnitt, wenn K endlich ist. Für jedes Ereignis $e \in E$, sind die Mengen ${}^\bullet e$ und $e^\bullet$ co-Mengen. Für zwei Schnitte C_1 und C_2 von K definieren wir: $C_1 \leq C_2$ genau dann, wenn $b_2 \not< b_1$ für alle $b_1 \in C_1$ und $b_2 \in C_2$ gilt.

Um einen verteilten Ablauf eines Systems zu modellieren, brauchen wir noch eine Funktion, die jedes Ereignis des Kausalnetzes als Schalten einer Transition des Systems identifiziert und jede Bedingung mit einer Stelle des Systems.

Definition 1.6 (Σ-Beschriftung)

Sei $\Sigma = ((P, T, F), M_0)$ ein System und $K = (B, E, \lessdot)$ ein Kausalnetz. Eine Funktion $r : B \cup E \to P \cup T$ ist eine *Σ-Beschriftung* von K, wenn die Funktion r Bedingungen auf Stellen und Ereignisse auf Transitionen abbildet. ⋆

[2] *co* steht abkürzend für das englische Wort *concurrent*

Durch eine Σ-Beschriftung können wir jeder endlichen co-Menge C von K kanonisch eine Markierung von Σ zuordnen. Wir bezeichnen diese Markierung mit $r(C)$ und definieren $r(C) : \mathcal{P} \to \mathbb{N}$ mit $r(C)(p) = |\{b \in C \mid r(b) = p\}|$.

Definition 1.7 (Prozeß)

Sei $\Sigma = ((P, T, F), M_0)$ ein System. Das Paar $\rho = (K, r)$ ist ein *Prozeß* von Σ, wenn $K = (B, E, \lessdot)$ ein Kausalnetz und $r : B \cup E \to P \cup T$ eine Σ-Beschriftung sind, so daß

(i) $r({}^\circ\mathrm{K}) = M_0$

(ii) für jedes Ereignis $e \in E$ und jede Transition $t \in T$ gilt: Wenn $r(e) = t$, dann ist

$$r({}^\bullet e) = t^- \text{ und}$$
$$r(e^\bullet) = t^+$$

$\star$

Mit Bedingung (i) fordern wir, daß der Anfangsschnitt auf die Anfangsmarkierung des Petrinetzes abgebildet wird. In Bedingung (ii) verlangen wir, daß jedes Ereignis e dem Schalten einer Transition t des Petrinetzes entspricht, d.h. e wird auf t abgebildet und der Vor- und Nachbereich von e auf den Vor- bzw. Nachbereich von t. Wir nennen e ein *Auftreten* der Transition t im Prozeß.

Sei ρ ein Prozeß und sei C eine co-Menge des Kausalnetzes, das ρ zugrunde liegt. Wenn wir alle Elemente des Kausalnetzes weglassen, die größer als C sind, dann erhalten wir ein *Anfangsstück* von ρ. Ein Prozeß ρ' ist ein Präfix des Prozesses ρ, wenn man das Kausalnetz von ρ' isomorph auf ein Anfangsstück von ρ abbilden kann, so daß die Beschriftungen mit denen von ρ übereinstimmen.

Definition 1.8 (Präfix eines Prozesses)

Sei $\Sigma = ((P, T, F), M_0)$ ein System und seien $\rho = ((B, E, \lessdot), r)$ und $\rho' = ((B', E', \lessdot'), r')$ Prozesse von Σ. ρ' ist ein Präfix von ρ, wenn es eine injektive Funktion $f : B' \cup E' \longrightarrow B \cup E$ gibt, so daß für alle $x \in B' \cup E'$:

(i) $\{f(y) \mid y \in {}^\bullet x\} = {}^\bullet(f(x))$ und

(ii) $r'(x) = r(f(x))$. $\star$

Verteilte Abläufe sind Prozesse, die maximal bzgl. der Präfixrelation sind. Wir definieren dies durch die *Progressannahme*, daß keine Transition von Σ in K° aktiviert ist. In sicheren Systemen ist diese Forderung äquivalent zu der Aussage: Jede aktivierte Transition schaltet irgendwann, falls keine Konflikttransition schaltet.

Definition 1.9 (Verteilter Ablauf)

Sei $\Sigma = ((P, T, F), M_0)$ ein System. Das Paar $\rho = (K, r)$ ist ein *verteilter Ablauf* (kurz: *Ablauf*) von Σ, wenn ρ ein Prozeß von Σ ist und für alle Transitionen t aus T und alle endlichen co-Mengen $C \subseteq \mathrm{K}^\circ$ gilt

$$t^- \not\leq r(C)$$

Die Menge aller verteilten Abläufe von Σ bezeichnen wir mit $\mathcal{R}(\Sigma)$. $\star$

Das System Σ_1 hat genau einen Ablauf, und dieser Ablauf ist unendlich.

Sei ρ ein verteilter Ablauf eines Systems Σ. Eine *Sequentialisierung* von ρ ist eine Folge von Schritten $M_0 \stackrel{r(e_1)}{\rightarrow} M_1 \stackrel{r(e_2)}{\rightarrow} M_2 \ldots$, in der jedes Ereignis von ρ genau einmal vorkommt und zwar so, daß die partielle Ordnung der Ereignisse respektiert wird, d.h. wenn $e_i < e_j$ im verteilten Ablauf gilt, dann ist $i < j$.

Bemerkung 1.10 Eine Markierung M von Σ ist genau dann erreichbar, wenn es einen verteilten Ablauf mit einem Schnitt C gibt, so daß $r(C) = M$. $\star$

1.3 Ersetzungsnetze und expandierte Abläufe

In diesem Abschnitt definieren wir einige Begriffe, die wir zum Verständnis und zur formalen Definition von Transitionsverfeinerung benötigen.

Der erste Begriff, den wir formal definieren, ist ein *Ersetzungsnetz* für eine Transition. Ein Ersetzungsnetz N ist ein Petrinetz, das die Stellen des Vor- und Nachbereichs der zu ersetzenden Transition t enthält und das sich mit der Anfangsmarkierung t^- wie t verhält. Das bedeutet, wenn wir als Anfangsmarkierung des Ersetzungsnetzes die Markierung t^- betrachten, in der auf allen Stellen aus dem Vorbereich von t genau eine Marke liegt und alle anderen Stellen unmarkiert sind, dann ist jeder Ablauf des Systems (N, t^-) endlich und im Endzustand liegt auf jeder Stelle des Nachbereichs von t genau eine Marke und alle anderen Stellen sind unmarkiert. Wir betrachten noch einmal das Beispiel in Abb. 1.10.

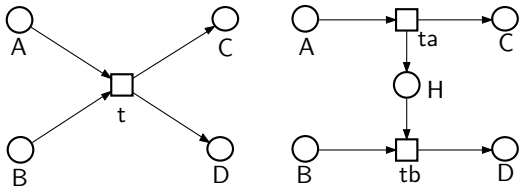


Abb. 1.10: Eine Transition t und das Ersetzungsnetz N

Wenn t schaltet, dann ändert sich die Markierung von A, B auf C, D. Wenn das Netz N in einem Zustand startet, in dem nur auf A und B jeweils eine Marke liegt, dann liegt im Endzustand auf C und D jeweils eine Marke.

Definition 1.11 (Ersetzungsnetz)

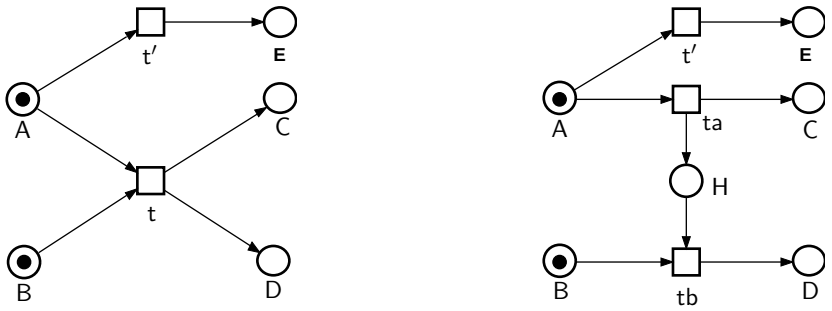
Sei $N = (P, T, F)$ ein Netz und sei $t \in T$ eine Transition von N. Ein Netz $N_E = (P_E, T_E, F_E)$ ist ein *Ersetzungsnetz* für t, wenn

(i) $P \cap P_E = {}^\bullet t \cup t^\bullet$,

(ii) $T \cap T_E = \emptyset$,

(iii) jeder Ablauf $\rho = (\mathrm{K}, r)$ von (N_E, t^-) ist endlich und endet mit $r(\mathrm{K}^\circ) = t^+$. $\star$

Wir haben verlangt, daß sich das Ersetzungsnetz mit der kanonischen Anfangsmarkierung t^- wie die Transition verhält. In dieser Definition ist es also nicht erlaubt,



(a) Σ_a (b) Σ'_a

Abb. 1.11: Ein System Σ_a und das erweiterte System Σ'_a

daß auf neuen Stellen des Ersetzungsnetzes (z.B. der Stelle H in Abb. 1.10) bereits zu Beginn Marken liegen. Wir diskutieren eine Abschwächung dieser Forderung im letzten Abschnitt dieser Arbeit (Seite 154).

In der nächsten Definition formalisieren wir, wie eine Transition durch ein Ersetzungsnetz ersetzt wird, was offensichtlich und rein technisch ist.

Definition 1.12 (Erweitertes Netz)

Sei $N = (P, T, F)$ ein Netz, sei t eine Transition des Netzes und sei $N_E = (P_E, T_E, F_E)$ ein Ersetzungsnetz für t. Das *erweiterte Netz* $N\big[t \to N_E\big]$ ist das Netz (P', T', F'), das gegeben ist durch $P' = P \cup P_E$, $T' = (T \setminus \{t\}) \cup T_E$ und $F' = (F \cup F_E) \cap (P' \times T' \cup T' \times P')$.

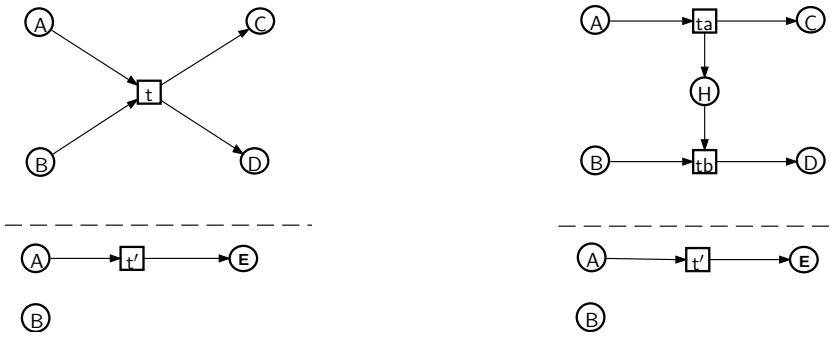
Für ein System $\Sigma = (N, M_0)$ bezeichnen wir mit $\Sigma\big[t \to N_E\big]$ das erweiterte System $(N\big[t \to N_E\big], M'_0)$, wobei $M'_0(p) = M_0(p)$ für $p \in P$ und $M'_0(p) = 0$ für alle anderen p. ⋆

Wir verweisen auf Σ ohne die Transition t als die *Umgebung* von t bzw. die *Umgebung* von N_E. Es hängt von der Umgebung ab, ob ein Ersetzungsnetz eine Verfeinerung einer Transition ist oder nicht. Von einer Verfeinerung verlangen wir, daß wichtige Eigenschaften des Systems erhalten werden. In verteilten Algorithmen interessieren uns Ablaufeigenschaften, also Eigenschaften die in jedem Ablauf des Systems gelten, wie "in jedem Ablauf gilt immer..." oder "in jedem Ablauf gilt irgendwann...". Deshalb verlangen wir, daß bei einer Verfeinerung das Ausgangssystem und das verfeinerte System ähnliche Abläufe haben.

Wir betrachten das System Σ_a mit der Transition t und das System Σ'_a, in dem t durch das Ersetzungsnetz N ersetzt wurde (Abb. 1.11). Die Abläufe des erweiterten Systems Σ'_a sind gerade die Abläufe des Ausgangssystems Σ_a, nur daß jedes Auftreten der Transition t durch einen Ablauf des Ersetzungsnetzes[3] ersetzt wurde (siehe Abb. 1.12). Den Ablauf $\rho_{a'1}$ erhalten wir aus ρ_{a1}, indem wir das mit t beschriftete

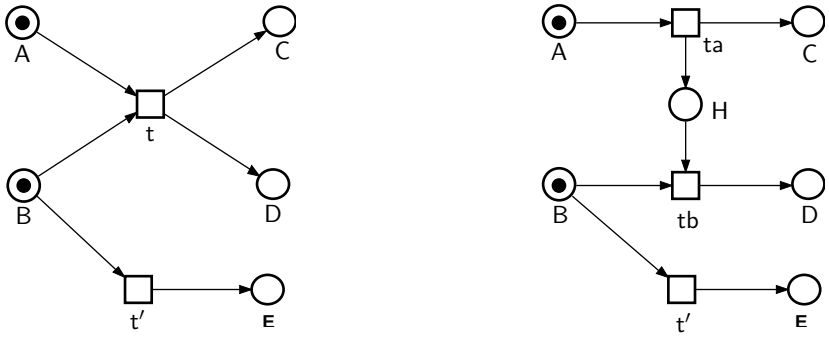
[3]In diesem und in den meisten anderen Beispielen in dieser Arbeit hat das jeweilige Ersetzungsnetz (bis auf Isomorphie der Kausalnetze) nur einen Ablauf. Das liegt daran, daß in unseren Beispielen innerhalb eines Ersetzungsnetzes keine Konflikte auftreten. Wenn ein Konflikt in einem Ersetzungsnetz auftritt, dann haben nach Definition trotzdem alle Abläufe des Ersetzungsnetzes den gleichen Endzustand, unabhängig davon, wie ein Konflikt entschieden wird.

Ereignis durch den Ablauf von N ersetzen. Den Ablauf $\rho_{a'2}$ erhalten wir ohne ρ_{a2} zu verändern, denn t tritt in ρ_{a2} nicht auf. Im System Σ_a ist das Ersetzungsnetz N deshalb eine Transitionsverfeinerung von t.



(a) ρ_{a1} und ρ_{a2} (b) $\rho_{a'1}$ und $\rho_{a'2}$

Abb. 1.12: Die verteilten Abläufe von Σ_a und Σ'_a



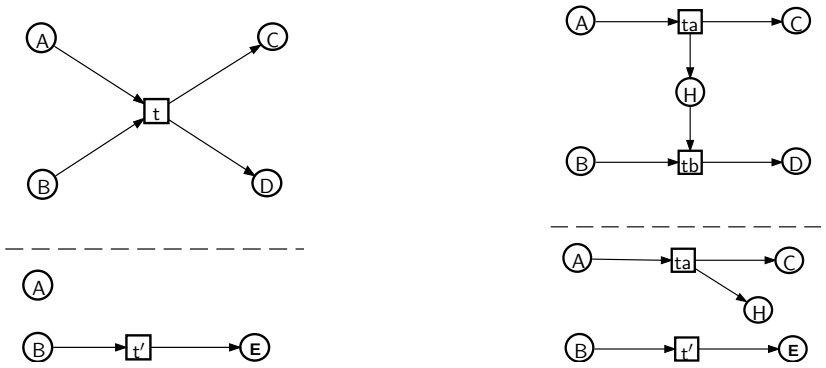
(a) Σ_b (b) Σ'_b

Abb. 1.13: Ein System Σ_b und das erweiterte System Σ'_b

Im System Σ_b (Abb. 1.13) ist N keine Transitionsverfeinerung von t, denn im Ablauf $\rho_{b'2}$ schalten ta und $\mathsf{t'}$. Der Ablauf endet in einem Zustand, in dem C und E markiert sind. Der Ablauf $\rho_{b'2}$ hat keine Entsprechung in einem Ablauf von Σ_b.

Bevor wir Transitionsverfeinerung formal definieren, brauchen wir noch einen Begriff, der die "Ähnlichkeit" von Abläufen ausdrückt. Wir nennen einen Ablauf des Ausgangsnetzes, in dem jedes Auftreten von t durch einen Ablauf des Ersetzungsnetzes ersetzt wurde eine $\big[t \to N_E\big]$-Expansion dieses Ablaufs (schematisch dargestellt in Abb 1.15). Wenn das Ersetzungsnetz mehrere Abläufe hat, dann können verschiedene Auftritte von t durch verschiedene Abläufe ersetzt werden. Nach der Definition des Ersetzungsnetzes haben alle Abläufe den gleichen Anfangszustand und den gleichen Endzustand.

Obwohl es intuitiv einfach ist, diese Ersetzungen in einem Ablauf durchzuführen, ist die Definition wegen der technischen Einzelheiten lang. Die Definition ist aber



(a) ρ_{b1} und ρ_{b2} (b) $\rho_{b'1}$ und $\rho_{b'2}$

Abb. 1.14: Die verteilten Abläufe von Σ_b und Σ'_b

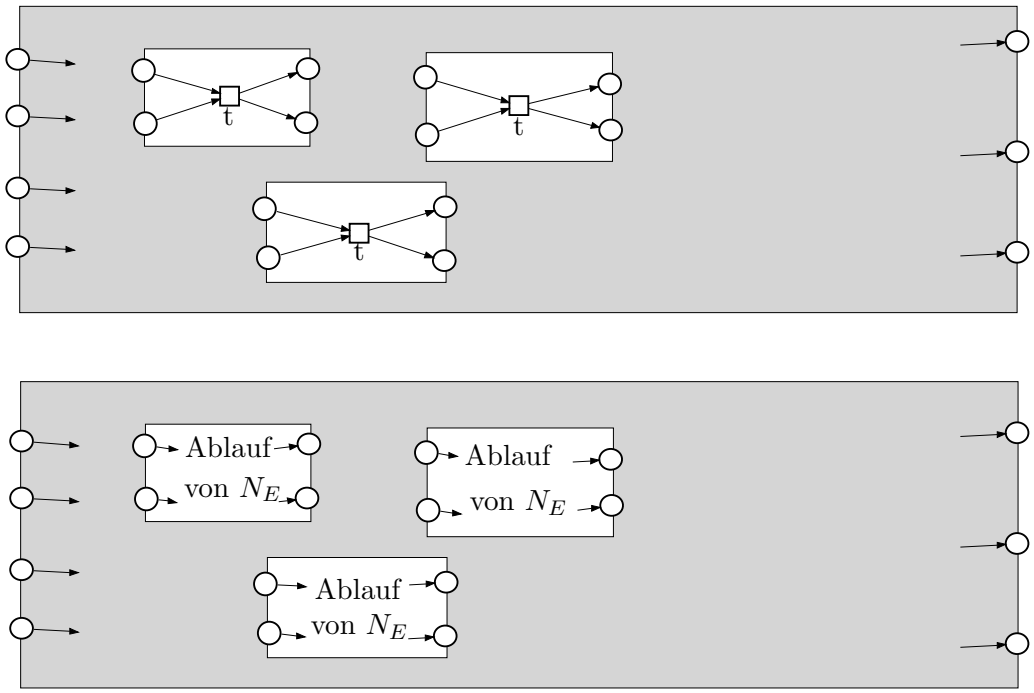


Abb. 1.15: Ein Ablauf und eine Expansion des Ablaufs

kanonisch, es gibt keinen Trick oder tiefere Einsichten. Der Aufwand der formalen Definition gegenüber der informellen liegt in der sorgfältigen (disjunkten) Bezeichnung der Bedingungen und Ereignisse, so daß ein Ablauf der Ersetzungsnetzes leicht in den Ablauf des Ausgangssystem eingefügt werden kann.

Definition 1.13 (Expansion eines Ablaufs)

Seien $\Sigma = ((P, T, F), M_0)$ ein System, t eine Transition von Σ, $N_E = (P_E, T_E, F_E)$ ein Ersetzungsnetz für t und $\rho = ((B, E, \lessdot), r)$ ein Ablauf von Σ. Wir definieren

(i) $E(t) = \{a \in E \mid r(a) = t\}$ als die Menge aller Ereignisse, die einem Auftreten

von t in ρ entsprechen,

(ii) für jedes $a \in E(t)$ einen Ablauf (K_a, r_a) von (N_E, t^-), so daß ${}^\circ\mathrm{K}_a = {}^\bullet a$ und $\mathrm{K}_a^\circ = a^\bullet$ und für alle $b \in {}^\bullet a \cup a^\bullet$ gilt $r(b) = r_a(b)$, so daß die Mengen $E_a \cup B_a \setminus ({}^\circ\mathrm{K}_a \cup \mathrm{K}^\circ{}_a)$ paarweise disjunkt für verschiedene a und außerdem paarweise disjunkt mit $E \cup B$ sind,

(iii) das Kausalnetz $(B', E', \lessdot')$, gegeben durch

$$\begin{aligned} B' =& B \cup \bigcup\{B_a \mid a \in E(t)\},\\ E' =& (E \setminus E(t)) \cup \bigcup\{E_a \mid a \in E(t)\},\\ \lessdot' =& \big(\lessdot \setminus (E(t) \times B \cup B \times E(t))\big) \cup \bigcup\{\lessdot_a \mid a \in E(t)\}, \end{aligned}$$

(iv) die $\Sigma\big[t \to N_E\big]$-Beschriftung r', gegeben durch

$$r'(x) = \begin{cases} r(x) & \text{für } x \in B \cup E \setminus E(t),\\ r_a(x) & \text{für } x \in B_a \cup E_a. \end{cases}$$

Dann nennen wir $\rho' = ((B', E', \lessdot'), r')$ eine $\big[t \to N_E\big]$-*Expansion* von ρ. ⋆

Nach dieser Definition ist eine $\big[t \to N_E\big]$-Expansion eines Ablaufs ρ von Σ ein Kausalnetz mit $\Sigma\big[t \to N_E\big]$-Beschriftung. Wir zeigen, daß sie sogar ein Prozeß von $\Sigma\big[t \to N_E\big]$ ist, aber kein Ablauf von $\Sigma\big[t \to N_E\big]$ sein muß.

Wir betrachten noch einmal den Ablauf ρ_{b2} von Σ_b, in dem nur t' auftritt (Abb. 1.14). Die Expansion von ρ bzgl. des Ersetzungsnetzes N aus Abb. 1.10 ist ρ_{b2} selbst, denn die Transition t tritt in ρ_{b2} nicht auf. Offensichtlich ist ρ_{b2} ein Prozeß von Σ'_b aber kein Ablauf, da die Transition ta (eine Transition des Ersetzungsnetzes) im Endschnitt von ρ_{b2} aktiviert ist.

Lemma 1.14 Sei $\rho' = (K', r')$ eine $\big[t \to N_E\big]$-Expansion eines Ablaufs $\rho = (K, r)$ von Σ. Dann ist ρ' ein Prozeß von $\Sigma\big[t \to N_E\big]$ und in K'° sind höchstens Transitionen aus N_E aktiviert. ⋆

Beweis: Durch die Konstruktion in Def. 1.13, ist ρ' ein Kausalnetz mit einer $\Sigma\big[t \to N_E\big]$-Beschriftung. Um zu beweisen, daß ρ' ein Prozeß von $\Sigma\big[t \to N_E\big]$ ist, müssen wir nur zeigen, daß die Bedingungen (i) und (ii) von Def. 1.7 gelten. Das folgt aber unmittelbar aus der Konstruktion einer $\big[t \to N_E\big]$-Expansion.

Durch diese Konstruktion wissen wir auch, daß die Enden K° und K'° eines Ablaufs und seiner Expansion übereinstimmen. In K° ist keine Transition von Σ aktiviert. In der Markierung $r'(\mathrm{K}'^\circ)$ von $\Sigma\big[t \to N_E\big]$ können also höchstens Transitionen aus N_E aktiviert sein. q.e.d.

Sei $\rho = (\mathrm{K}, r)$ ein Ablauf eines Systems Σ. Eine Stelle p von Σ heißt *persistent* in ρ, wenn p am Ende von ρ markiert ist, also wenn $p \in r(\mathrm{K}^\circ)$.

Folgerung 1.15 Sei ρ ein Ablauf von Σ. Wenn keine Stelle aus dem Vorbereich von t persistent ist, dann ist jede $\big[t \to N_E\big]$-Expansion von ρ ein Ablauf von $\Sigma\big[t \to N_E\big]$. ⋆

Mit Hilfe aller Zutaten ist es nun einfach, Transitionsverfeinerung zu definieren.

Definition 1.16 (Elementare Transitionsverfeinerung)

Sei Σ ein System und sei t eine Transition von Σ. N_E ist eine *elementare Transitionsverfeinerung* der Transition t im System Σ, wenn N_E ein Ersetzungsnetz ist und die Menge aller Abläufe von $\Sigma\big[t \to N_E\big]$ gleich der Menge aller $\big[t \to N_E\big]$-Expansionen von Abläufen von Σ ist. $\star$

Nach dieser Definition ist N aus Abb. 1.10 eine elementare Transitionsverfeinerung von `t` in Σ_a, aber es ist keine Transitionsverfeinerung in Σ_b. Später definieren wir noch *algebraische Transitionsverfeinerung*. Wir benutzen das Wort *Transitionsverfeinerung* ohne Adjektiv, wenn aus dem Zusammenhang hervorgeht, ob es sich um algebraische oder elementare Transitionsverfeinerung handelt.

2 Beweiskriterien für elementare Transitionsverfeinerung

Es ist im allgemeinen schwer zu zeigen, daß ein Ersetzungsnetz eine Transitionsverfeinerung ist, wenn man nur die Definition 1.16 zur Verfügung hat. Wir beleuchten den Begriff Transitionsverfeinerung von verschiedenen Seiten, um zu verstehen, was man eigentlich beweisen muß und geben eine äquivalente Charakterisierung für Transitionsverfeinerung an. Dann geben wir einfache Beweiskriterien für verschiedene Klassen von Systemen an.

2.1 Ein semantisches Kriterium

Das folgende Lemma vereinfacht die Beweisobligation für Transitionsverfeinerung bereits, indem aus dem "ist gleich" in Definition 1.16 ein "ist enthalten in" wird.

Lemma 2.1 Wenn N_E ein Ersetzungsnetz für die Transition t ist und jeder Ablauf von $\Sigma\big[t \to N_E\big]$ eine $\big[t \to N_E\big]$-Expansion eines Ablaufs von Σ ist, dann gilt auch die umgekehrte Inklusion: jede $\big[t \to N_E\big]$-Expansion eines Ablaufs von Σ ist ein Ablauf von $\Sigma\big[t \to N_E\big]$. $\star$

Es ist vielleicht ein wenig überraschend, daß eine Mengeninklusion aus ihrer umgekehrten Inklusion folgt. Die Gründe dafür sind einfach:

Beweis: (Lemma 2.1)
Sei ρ ein Ablauf von Σ und ρ' eine $\big[t \to N_E\big]$-Expansion von ρ. Wir müssen zeigen, daß ρ' ein Ablauf von $\Sigma\big[t \to N_E\big]$ ist.

Durch Lemma 1.14 wissen wir schon, daß ρ' ein Prozeß von $\Sigma\big[t \to N_E\big]$ ist. Wir setzen ρ' zu einem Ablauf σ' von $\Sigma\big[t \to N_E\big]$ fort. Laut Voraussetzung ist σ' eine $\big[t \to N_E\big]$-Expansion eines Ablaufs σ von Σ. Nach unserer Konstruktion ist ρ ein Präfix von σ. Da jeder Ablauf ein maximaler Prozeß ist, folgt $\rho = \sigma$. Folglich ist $\rho' = \sigma'$ und ρ' ist ein Ablauf von $\Sigma\big[t \to N_E\big]$, da σ' ein Ablauf ist. q.e.d.

Wir betrachten noch einmal das System Σ_1' von Seite 18. In einem sequentiellen Ablauf von Σ_1' müssen die zu demselben Ablauf des Ersetzungsnetzes gehörenden Ereignisse von ta und tb nicht direkt hintereinander auftreten. Zwischen den beiden Ereignissen könnte die Transition $t4$ liegen. Wenn man die sequentiellen Abläufe des Systems betrachtet, geht die Atomarität einer Transition meist verloren, wenn die Transition durch ein Teilsystem verfeinert wird. Der folgende Satz besagt, daß es bei einer Transitionsverfeinerung immer noch eine Art Atomarität gibt, die wir

in einem verteilten Ablauf sichtbar machen können: Wir können jeden Ablauf des Ersetzungsnetzes innerhalb eines Ablaufs des verfeinerten Netzes mit zwei Schnitten herausschneiden, so daß zwischen den beiden Schnitten genau die Ereignisse des Ersetzungsnetzes liegen.

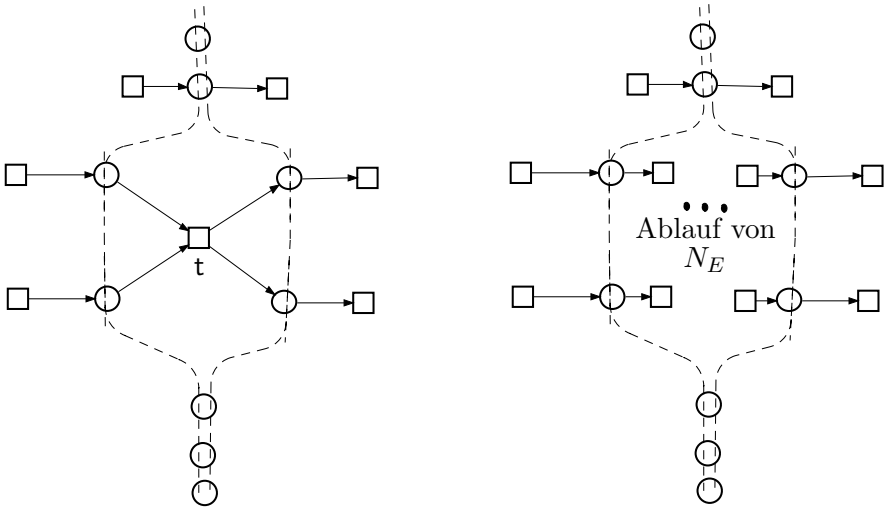


Abb. 2.1: Illustration für Satz 2.2

Dies ist in Abb. 2.1 illustriert. Auf der linken Seite ist ein Teil eines Ablaufs des Ausgangssystems mit einem Auftreten der Transition t dargestellt. Die gestrichelten Linien sind Schnitte zwischen denen genau das Auftreten der Transition t liegt. Solche Schnitte findet man zu jedem beliebigen Ereignis, da jedes Ereignis atomar ist, aber die Schnitte sind im allgemeinen nicht eindeutig bestimmt. Der Satz besagt nun, daß in einem Ablauf des verfeinerten Systems alle Ereignisse des Ersetzungsnetzes zu Abläufen des Ersetzungsnetzes geordnet werden können und daß diese Abläufe atomar sind in dem Sinne, daß der Ablauf durch zwei Schnitte aus dem gesamten Ablauf des verfeinerten Systems herausgeschnitten werden kann. Das bedeutet auch, daß jeder Ablauf des verfeinerten Systems eine Sequentialisierung hat, in der alle Ereignisse eines Ablaufs des Ersetzungsnetzes hintereinander auftreten.

Bevor wir das Theorem formulieren, legen wir noch einige Notationen fest. Sei $K = (B, E, \lessdot)$ ein Kausalnetz und C ein Schnitt in K. Da ein Schnitt eine maximale co-Menge ist, gibt es zu jedes Ereignis $e \in E$ eine Bedingung $b \in C$, so daß entweder $e < b$ oder $e > b$. Wir schreiben dann $e < C$ bzw. $e > C$. Für eine Teilmenge $E' \subseteq E$ schreiben wir $E' < C$, wenn für alle $e \in E'$ gilt $e < C$. Analog benutzen wir auch $E' > C$.

Eine *Partition* einer Menge A ist eine Menge $\{A_i \mid i \in$ einer Indexmenge$\}$ paarweise disjunkter, nicht leerer Teilmengen von A, so daß $\bigcup_i A_i = A$. Jede Menge A_i nennen wir eine *Äquivalenzklasse* der Partition.

Satz 2.2 Sei N_E ein Ersetzungsnetz für die Transition t im System Σ. N_E ist genau dann eine Transitionsverfeinerung, wenn zu jedem Ablauf $\rho' = ((B', E', \lessdot'), r')$ von $\Sigma\big[t \to N_E\big]$ eine Partition der Menge $\{e \in E' \mid r'(e) \in T_E\}$ aller N_E-Ereignisse des Ablaufs existiert, so daß für jede Äquivalenzklasse A der Partition gilt:

(i) Es gibt zwei Schnitte C_1, C_2 in ρ', so daß

(a) die Ereignisse zwischen diesen Schnitten genau die Ereignisse der Äquivalenzklasse A sind, d.h. $C_1 \leq C_2$ und für alle $e \in E'$ gilt

$$C_1 < e < C_2 \qquad \Longleftrightarrow \qquad e \in A.$$

(b) Für jede Äquivalenzklasse $B \neq A$ gilt: entweder $B < C_1$ oder $B > C_2$. Das bedeutet, daß die Schnitte C_1, C_2 keine anderen Äquivalenzklassen zerschneiden.

(ii) Das Teilnetz, das durch die Ereignisse der Äquivalenzklasse A induziert[1] wird, ist zusammen mit der Restriktion von r' auf diese Ereignisse ein Ablauf von (N_E, t^-). $\star$

Beweis: ($\Longrightarrow$) Diese Implikation folgt direkt aus den Definitionen 1.16 und 1.13. ($\Longleftarrow$) Sei $\rho' = ((B', E', \prec'), r')$ ein beliebiger Ablauf von $\Sigma\big[t \to N_E\big]$. Angenommen, es existiert eine oben beschriebene Partition. Nach Lemma 2.1 müssen wir zeigen, daß es einen Ablauf ρ von (Σ, M_0) gibt, so daß ρ' eine $\big[t \to N_E\big]$-Expansion von ρ ist.

Wir konstruieren ρ, indem wir jedes durch eine Äquivalenzklasse induzierte Teilnetz von ρ' durch ein mit t beschriftetes Ereignis ersetzen. Wir überprüfen die Bedingungen aus Def. 1.9 und Def. 1.13 und stellen fest, daß ρ tatsächlich ein Ablauf von Σ ist und ρ' eine $\big[t \to N_E\big]$-Expansion von ρ ist. q.e.d.

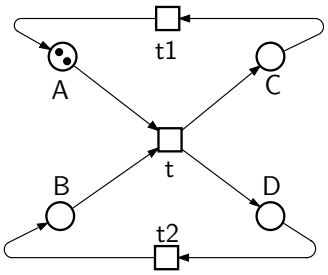
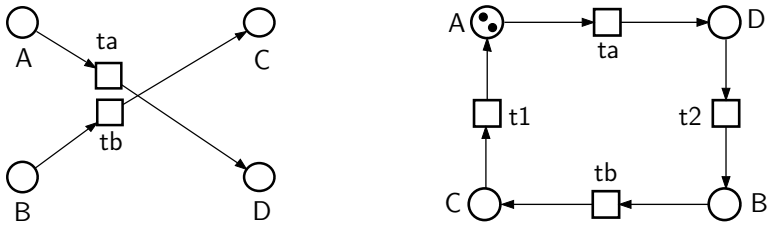


Abb. 2.2: Das System Σ

Im nächsten Abschnitt zeigen wir, daß wir die Bedingung (i)(b) weglassen können, wenn $\Sigma\big[t \to N_E\big]$ sicher ist. Das folgende Beispiel zeigt, daß diese Bedingung im allgemeinen aber notwendig ist. Im System Σ (Abb. 2.2) kann keine Transition schalten. Wenn wir t durch zwei zueinander nebenläufige Transitionen (Abb. 2.3(a)) ersetzen, erhalten wir das System $\Sigma\big[t \to N_E\big]$ (Abb. 2.3(b)).

Das System $\Sigma\big[t \to N_E\big]$ hat nur einen verteilten Ablauf, in dem jede Transition unendlich oft schaltet. Das Ersetzungsnetz N_E ist also keine Transitionsverfeinerung von t in Σ. Wir können die Transitionen des Ersetzungsnetzes in diesem Ablauf aber so partitionieren, daß alle Bedingungen von Satz 2.2 außer Bedingung (iii) erfüllt sind. Wir finden die geforderten Schnitte, allerdings nur so, daß die Schnitte einer

[1] Das von einer Menge von Ereignissen induzierte Netz besteht genau aus den Ereignissen der Menge, ihren Vor- und Nachbereichen und den Kanten zwischen ihnen.



(a) N_E (b) $\Sigma[t \to N_E]$

Abb. 2.3: Ersetzungsnetz und erweitertes System

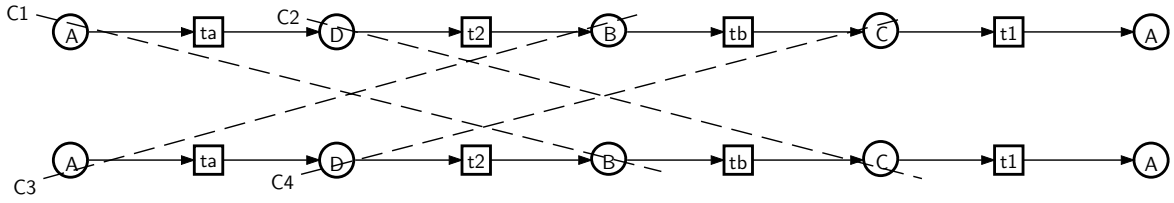


Abb. 2.4: Präfix des verteilten Ablaufs von $\Sigma[t \to N_E]$

Äquivalenzklasse eine andere Äquivalenzklasse zerschneiden (siehe Abb. 2.4). Die Schnitte $C1$ und $C2$ grenzen einen Ablauf des Ersetzungsnetzes ein, genau wie die Schnitte $C3$ und $C4$.

2.2 Beweiskriterien für sichere Systeme

Wenn eine ersetzte Transition nebenläufig zu sich selbst schalten kann, ist ein Ersetzungsnetz oft keine Transitionsverfeinerung, weil zwei verschiedene Abläufe des Ersetzungsnetzes durcheinander geraten können. Wir beschränken uns nun auf sichere Systeme, in denen solche Selbstnebenläufigkeit nicht vorkommt. Diese Einschränkung ist praktisch nicht relevant, denn so gut wie jeder verteilte Algorithmus kann als sicheres System modelliert werden. Wir gehen darauf noch genauer im Abschnitt 4.1 ein. Sichere Systeme sind ausführlich theoretisch untersucht worden. Es gibt in den meisten Fällen einfache syntaktische Kriterien, um zu zeigen, daß ein System sicher ist. Für ein weiterführendes Studium sicherer Systeme siehe z.B. [Rei85].

Satz 2.2 ist eine äquivalente Charakterisierung des Begriffs Transitionsverfeinerung. Wenn wir eine Partition gefunden haben, ist es meist einfach zu zeigen, daß die Bedingung (ii) des Satzes gilt. Auch die Bedingung (i)(a) ist oft einfach beweisbar, da wir nur eine Äquivalenzklasse betrachten müssen. Meist ist es jedoch schwer zu zeigen, daß Bedingung (i)(b) gilt, denn wir müssen alle Äquivalenzklassen gleichzeitig betrachten. Diese schwierige Beweisobligation zeigen wir nun für alle sicheren Systeme auf einmal. Wir beweisen, daß wenn eine Partition existiert, die alle Bedingungen von Satz 2.2 außer Bedingung (i)(b) erfüllt, dann existiert auch eine Partition, die

alle Bedingungen dieses Satzes erfüllt. Der Beweis ist technisch sehr aufwendig und wird deshalb im Anhang A.1 angegeben. Wir geben hier nur die Beweisidee an.

Satz 2.3 Sei N_E ein Ersetzungsnetz für die Transition t im System Σ und sei $\Sigma\big[t \to N_E\big]$ sicher. Wenn es zu jedem Ablauf von $\Sigma\big[t \to N_E\big]$ eine Partition der Menge $\{e \in E' \mid r'(e) \in T_E\}$ aller N_E-Ereignisse des Ablaufs gibt, die die Bedingungen (i)(a) und (ii) aus Satz 2.2 erfüllt, dann existiert für jeden Ablauf auch eine solche Partition, die außerdem die Bedingung (i)(b) aus Satz 2.2 erfüllt. ⋆

Beweisidee: Sei $\rho' = ((B', E', \prec'), r')$ ein Ablauf von $\Sigma\big[t \to N_E\big]$ und sei $\wp$ eine Partition aller N_E-Ereignisse von ρ', die die Bedingungen (i)(a) und (ii) aus Satz 2.2, aber nicht Bedingung (i)(b) erfüllt.

Seien A und B zwei beliebige verschiedene Äquivalenzklassen aus $\wp$ und seien C_1, C_2 zwei Schnitte, die die Bedingungen (i)(a) und (ii) für A erfüllen und D_1, D_2 die dementsprechend zu B gehörenden Schnitte. Wir beweisen zuerst, daß wir die Ereignisse aus $A \cup B$ so zu zwei neuen Äquivalenzklassen $\tilde{A}$ und $\tilde{B}$ umordnen können, daß es zu $\tilde{A}$ Schnitte gibt, die (i)(a) und (ii) erfüllen, und die $\tilde{B}$ nicht zerschneiden, und umgekehrt, daß es zu $\tilde{B}$ Schnitte gibt, die (i)(a) und (ii) erfüllen, und die $\tilde{A}$ nicht zerschneiden.

Danach zeigen wir, wie wir nach und nach je zwei Äquivalenzklassen auf diese Art umordnen, bis wir eine neue Partition erhalten, die alle Bedingungen von Satz 2.2 erfüllt. q.e.d.

Mit den Sätzen 2.2 und 2.3 erhalten wir eine äquivalente Charakterisierung des Begriffs Transitionsverfeinerung für sichere Systeme. Um zu zeigen, daß eine Ersetzung eine Verfeinerung ist, müssen wir also in jedem Ablauf des verfeinerten Systems die Ereignisse des Ersetzungsnetzes partitionieren. Wenn wir zum Beispiel eine Transition t durch zwei zueinander nebenläufige Transitionen t_1 und t_2 ersetzen, dann müssen wir nur noch beweisen: Immer wenn t_1 auftritt, tritt nebenläufig dazu genau einmal t_2 auf und umgekehrt.

Wir benutzen nun Satz 2.3, um einfache Beweiskriterien für spezielle Klassen von Systemen abzuleiten. Wir nehmen bei den folgenden Beweiskriterien immer an, daß Σ ein System, t eine Transition von Σ und $N_E = (P_E, T_E, F_E)$ ein Ersetzungsnetz für t ist, und daß Σ sicher ist. Wir nennen die Transitionen aus N_E, die Stellen aus dem Vorbereich von t im Vorbereich haben *Anfangstransitionen* des Ersetzungsnetzes und die Transitionen aus N_E, die Stellen aus dem Nachbereich von t im Nachbereich haben *Endtransitionen*.

Nun geben wir ein Kriterium dafür an, wann ein Ersetzungsnetz unabhängig von der Umgebung eine Transitionsverfeinerung ist. Das ist der Fall, wenn N_E keine verteilte Eingabe erlaubt, d.h. für alle Anfangstransitionen t_a gilt ${}^\bullet t_a = {}^\bullet t$. In diesem Fall ist das Ersetzungsnetz ein *Modul* im Sinne von Vogler [Vog87]. Vogler definiert ein Modul als Ersetzungsnetz, das in jeder möglichen Umgebung eine Verfeinerung ist. In einem Modul ist keine verteilte Eingabe erlaubt. Vogler entwickelt mit den Modulen eine Idee von Valette [Val79] weiter. Die Verfeinerung mit Modulen ist im Gegensatz zu Transitionsverfeinerung umgebungsunabhängig.

Satz 2.4 Sei Σ sicher, t eine Transition aus Σ und N_E ein Ersetzungsnetz für t. Wenn für alle Anfangstransitionen t_a von N_E gilt: ${}^\bullet t_a = {}^\bullet t$, dann ist N_E eine Transitionsverfeinerung von t in Σ. ⋆

Beweis: Wir betrachten ein sicheres System $\Sigma = (N, M_0)$ mit $N = (P, T, F)$, eine Transition $t \in T$ und ein Ersetzungsnetz $N_E = (P_E, T_E, F_E)$ für t. Sei T_A die Menge der Anfangstransitionen von N_E. Wir halten einen beliebigen Ablauf $\rho = (K, r)$ von $\Sigma\big[t \to N_E\big]$ fest. Wir müssen zeigen, daß es einen Ablauf σ von Σ gibt, so daß ρ eine $\big[t \to N_E\big]$-Expansion von σ ist.

Dazu beweisen wir folgendes: Wenn eine Anfangstransition von $\Sigma\big[t \to N_E\big]$ schaltet, dann wird erst der Ablauf des Ersetzungsnetzes zu Ende geführt, bevor wieder eine Anfangstransition aktiviert ist. Zum Beweis dieser Behauptung konstruieren wir σ aus ρ durch eine Art Simulation: Der Anfangsschnitt von σ ist gleich dem Anfangsschnitt von ρ. Wir betrachten nacheinander (so daß die Halbordnung der Ereignisse in ρ respektiert wird) die Ereignisse von ρ. Wenn eine Transition $t_1 \in T \setminus T_E$ auftritt, dann setzen wir den bereits konstruierten Präfix von σ mit einem Auftreten von t_1 fort. Wenn eine Anfangstransition t_A des Ersetzungsnetzes auftritt, dann setzen wir den Präfix von σ mit t fort. Wenn $t_E \in T_E \setminus T_A$ in ρ auftritt, dann lassen wir den Präfix von σ unverändert. Wir zeigen nun, daß diese Simulation funktioniert, also daß Σ immer wie gefordert schalten kann und σ wirklich ein Ablauf von Σ ist.

Dazu gehen wir noch einmal zu den Anfangsschnitten von ρ und σ zurück. Solange nur Transitionen aus $T \setminus T_E$ schalten, kann Σ jedes Ereignis von ρ simulieren und die Markierungen von Σ und $\Sigma\big[t \to N_E\big]$ sind vor und nach jedem Schalten gleich.

Wir betrachten nun einen Schnitt C in ρ, in dem zum ersten Mal das Auftreten einer Anfangstransition im Nachbereich von C ist, d.h. vor C tritt noch keine Anfangstransition auf und es gibt ein e mit ${}^\bullet e \subseteq C$ und $r(e) \in T_A$. Wir nehmen an, wir haben σ bereits bis zu dem zu C assoziierten Schnitt C' als Simulation von ρ konstruiert (siehe Abb. 2.5).

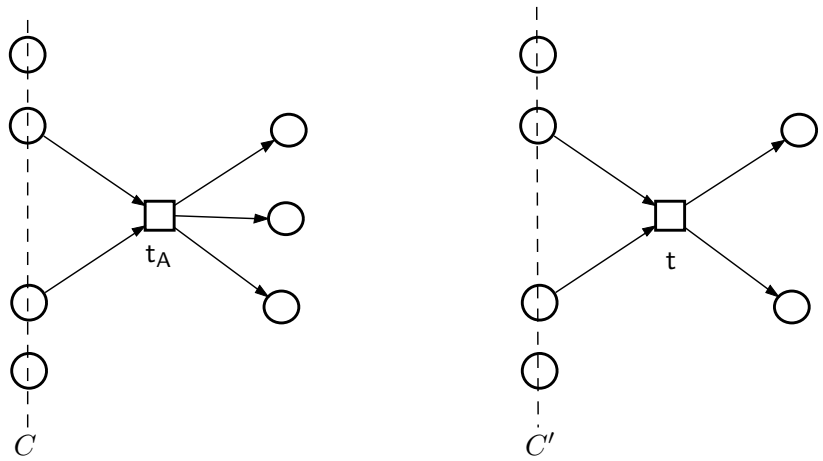


Abb. 2.5: Die Schnitte C in ρ und C' in σ

Zuerst stellen wir fest, daß höchstens eine Anfangstransition im Nachbereich von C liegen kann, da $r(C) = r'(C')$ und Σ sicher ist. Wir setzen den Präfix von σ fort, indem wir t schalten lassen. Nun setzen wir weiter fort wie vorher beschrieben: Wenn $t_1 \in T \setminus T_E$ in ρ auftritt, dann ist t_1 auch im Endschnitt vom bereits konstruierten Präfix von σ aktiviert und wir setzen den Präfix mit t_1 fort. Wenn $t_E \in T_E \setminus T_A$ auftritt, verändern wir den Präfix von σ nicht.

Wir müssen nun noch ausschließen, daß in ρ wieder eine Anfangstransition von N_E auftritt, solange der erste Ablauf des Ersetzungsnetzes noch nicht abgeschlossen ist. Wir nehmen also an, daß nach dem Schnitt C bisher nur Transitionen aus $T \setminus T_A$ aufgetreten sind und daß wir nun bei einem Schnitt C_1 angekommen sind, in dem der erste Ablauf des Ersetzungsnetzes noch nicht abgeschlossen ist, aber schon wieder eine Anfangstransition aktiviert ist (d.h. $t^- \leq r(C_1)$). Bis dahin konnten wir den Ablauf auch in σ wie beschrieben simulieren und haben nun einen Präfix von σ mit dem Endschnitt C_1' konstruiert, in dem nach Konstruktion $r(C_1)(p) = r(C_1')(p)$ für alle $p \in P \setminus P_A$ und alle $p \in {}^\bullet t$ und $r(C_1)(p) \leq r(C_1')(p)$ für alle $p \in t^\bullet$ gilt.

Da der Ablauf des Ersetzungsnetzes noch nicht abgeschlossen ist, sind zwischen C und C_1 noch nicht alle Marken aus dem Nachbereich von t erzeugt worden, also ist insbesondere noch nicht jede Marke verbraucht worden. Da in σ zwischen C' und C_1' die Transition t auftritt, sind bereits alle Stellen aus dem Nachbereich von t erzeugt, aber noch nicht wieder verbraucht. Da in C_1 eine Anfangstransition aktiviert ist, haben wir mit C_1' einen Zustand von Σ erreicht, in dem t aktiviert ist, aber mindestens eine Stelle aus dem Nachbereich von t markiert ist. Das widerspricht der Voraussetzung, daß Σ sicher ist.

Damit haben wir folgendes gezeigt: Wenn das erste Mal eine Anfangstransition von N_E in ρ auftritt, dann wird der mit diesem Ereignis begonnene Ablauf von N_E beendet, bevor das zweite Mal eine Anfangstransition aktiviert wird.

Induktiv können wir mit derselben Argumentation wie beim ersten Mal folgern, daß auch der i-te Ablauf des Ersetzungsnetzes beendet wird, bevor zum i+1-ten Mal eine Anfangstransition auftritt. Nach Satz 2.2 und Satz 2.3 ist damit N_E eine Transitionsverfeinerung von t in Σ. q.e.d.

Folgerung 2.5 Wenn Σ sicher ist und $|{}^\bullet t| = 1$ ist, dann ist jedes Ersetzungsnetz für t eine Transitionsverfeinerung von t in Σ. $\star$

Wenn $|{}^\bullet t| = 1$, dann synchronisert diese Transition nicht. Deshalb kann auch das Ersetzungsnetz nicht synchronisieren und verhält sich damit in bezug auf die Umgebung genauso wie die Transition. Diese einfache Folgerung ist praktisch relevant: ein Workflow-Netz [vdA98] hat genau eine Eingabe- und eine Ausgabestelle. Wenn man also eine einfache Transition, die nur eine Eingabestelle hat, durch ein Workflow-Netz ersetzt, dann ist das eine Transitionsverfeinerung, falls das Workflow-Netz ein Ersetzungsnetz ist. Allgemeiner können wir daraus schließen, daß Transitionsverfeinerung sehr einfach ist, wenn eine Transition durch das Ersetzungsnetz nur detaillierter beschrieben wird, ohne neue Konflikte mit der Umgebung zu erzeugen oder mehr Nebenläufigkeit in das System (durch Nebenläufigkeit im Ersetzungsnetz) einzuführen. Dieser Fall ist auch in anderen Formalismen in ähnlicher Weise beschreibbar. Interessant wird Transitionsverfeinerung erst, wenn verteilte Eingabe erlaubt ist und dadurch mehr Nebenläufigkeit oder mehr Konflikte in das System eingeführt werden.

Wir zeigen nun ein einfaches Beweiskriterium für den Fall, daß zwar verteilte Eingabe erlaubt ist, aber keine Konflikte im System auftreten.

Satz 2.6 Seien Σ ein System, t eine Transition aus Σ und N_E ein Ersetzungsnetz für t. Wir nehmen an, daß sowohl Σ, als auch (N_E, t^-) stellenunverzweigt und sicher

sind und außerdem keine Stelle des Vorbereichs von t in irgendeinem Ablauf von Σ persistent ist. Wenn auch $\Sigma\big[t \to N_E\big]$ sicher ist, dann ist N_E eine Transitionsverfeinerung von t in Σ. $\star$

Beweis: Jedes sichere stellenunverzweigte Netz hat (bis auf Isomorphie der Kausalnetze) genau einen Ablauf. Sei ρ der Ablauf von Σ. Da auch (N_E, t^-) nur einen Ablauf hat, hat ρ nur eine $\big[t \to N_E\big]$-Expansion ρ'. Diese ist nach Folgerung 1.15 ein Ablauf von $\Sigma\big[t \to N_E\big]$. Da auch $\Sigma\big[t \to N_E\big]$ sicher und stellenunverzweigt ist, ist ρ' (bis auf Isomorphie) der einzige Ablauf von $\Sigma\big[t \to N_E\big]$ und damit ist N_E eine Transitionsverfeinerung. q.e.d.

Wir geben nun ein Kriterium an, in dem wir verteilte Eingabe und Konflikte zulassen, aber die Konflikte zwischen Ersetzungsnetz und Umgebung einschränken. Wir fordern, daß eine Transition der Umgebung, die mit einer Anfangstransition des Ersetzungsnetzes in Konflikt steht, gleichzeitig mit allen Anfangstransitionen des Ersetzungsnetzes in Konflikt steht. Durch syntaktische Kriterien erreichen wir, daß ein Ablauf des Ersetzungsnetzes erst starten kann, wenn *alle* Stellen aus dem Vorbereich von t markiert sind. Da das System sicher ist, erreichen wir damit, daß ein Konflikt mit dem Ersetzungsnetz nur auftreten kann, bevor der Ablauf des Ersetzungsnetzes gestartet wurde. Während das Ersetzungsnetz arbeitet, ist keine konfliktäre Transition mehr aktiviert.

Wir nennen eine Transition t *schlingenfrei*, wenn ${}^\bullet t \cap t^\bullet = \emptyset$.

Satz 2.7 Sei Σ sicher, t eine schlingenfreie Transition aus Σ und N_E ein Ersetzungsnetz für t. N_E ist eine Transitionsverfeinerung von t, wenn die folgenden Bedingungen gelten:

(i) Das System $\Sigma\big[t \to N_E\big]$ ist sicher.

(ii) $T_E^\bullet \cap {}^\bullet t = \emptyset \qquad$ und $\qquad {}^\bullet T_E \cap t^\bullet = \emptyset$
(d.h. keine Transition aus T_E legt Marken in den Vorbereich von t oder benötigt zum Schalten Marken aus dem Nachbereich von t),

(iii) Sei $T_1 = \{t_1 \in T_E \mid {}^\bullet t_1 \cap {}^\bullet t \neq \emptyset\}$ die Menge der Anfangstransitionen von N_E. Wir fordern für alle $t_1, t_2 \in T_1$:

(a) $|{}^\bullet t_1 \cap {}^\bullet t| = |{}^\bullet t_2 \cap {}^\bullet t| \quad$ und $\quad \sum_{p \in {}^\bullet t_1} M_0(p) = \sum_{p \in {}^\bullet t_2} M_0(p)$
(d.h. alle Anfangstransitionen haben gleichviele Stellen aus dem Vorbereich von t im Vorbereich und in der Anfangsmarkierung ist die Summe der Marken auf diesen Plätzen für jede Anfangstransition gleich.)

(b) Für jede Transition $t' \in T$ gilt
$|t'^\bullet \cap {}^\bullet t_1 \cap {}^\bullet t| = |t'^\bullet \cap {}^\bullet t_2 \cap {}^\bullet t|$
(d.h. eine Transition der Umgebung von N_E legt die gleiche Anzahl von Marken in jeden Vorbereich einer Anfangstransition von N_E.)

(c) In $\Sigma\big[t \to N_E\big]$ gilt: wenn t_1 oder t_2 in Konflikt mit t' steht in einer erreichbaren Markierung M, dann ist
$|{}^\bullet t' \cap {}^\bullet t_1| = |{}^\bullet t' \cap {}^\bullet t_2|$
(d.h. eine Transition, die in Konflikt mit einer Anfangstransition von N_E steht, steht in Konflikt zu allen Anfangstransitionen und benötigt aus

dem Vorbereich jeder Anfangstransition die gleiche Anzahl von Marken.)
$\star$

Die Bedingungen (ii) und (iii)(a) und (b) sind einfache syntaktische Kriterien, die man an der Netzstruktur ablesen kann. Die Bedingungen (i) und (iii)(c) können mit Hilfe spezieller Invarianten (Stelleninvarianten, siehe Abschnitt 4.4) bewiesen werden, die syntaktisch überprüfbar sind.

Beweis: (von Satz 2.7) Wir beweisen diesen Satz, indem wir die Sätze 2.2 und 2.3 benutzen. Wir zeigen, daß alle Kriterien von Satz 2.2 erfüllbar sind, indem wir zeigen, daß die Bedingungen (i)(a) und (ii) erfüllbar sind. Dann können wir mit Satz 2.3 folgern, daß zusätzlich auch Bedingung (i)(b) erfüllbar ist.

Sei $\rho = ((B, E, \lessdot), r)$ ein Ablauf von $\Sigma\big[t \to N_E\big]$. Zuerst stellen wir fest, daß keine Transition von $\Sigma\big[t \to N_E\big]$ nebenläufig zu sich selbst in ρ auftreten kann, denn $\Sigma\big[t \to N_E\big]$ ist sicher. Folglich sind für jede Transition $t' \in T_E$ alle Ereignisse, die mit t' beschriftet sind total geordnet durch die transitive Hülle von $\lessdot$. Wir definieren E_i als die Menge aller Ereignisse e, die das i-te Auftreten einer Transition aus T_E in ρ anzeigen.

Offensichtlich bilden die nicht leeren Mengen E_i zusammen eine Partition der Ereignisse des Ersetzungsnetzes in ρ. Wir müssen nun zeigen, daß die Bedingungen (i) und (ii) von Satz 2.2 für diese Partition gelten.

Angenommen, es gibt ein i, so daß E_i nicht leer ist und das die Bedingungen (i) und (ii) von Satz 2.2 nicht erfüllt. Sei i der kleinste Index mit dieser Eigenschaft.

Behauptung 1: Es gibt einen Schnitt C, so daß

(i) es gibt ein $e \in E_i$ mit ${}^\bullet e \subseteq C$ und

(ii) für alle $e' \in E_i$ gilt $C < e'$ und

(iii) wenn $i > 1$ dann gilt $e' < C$ für alle $e' \in E_{i-1}$.

Die Behauptung gilt, da $\Sigma\big[t \to N_E\big]$ sicher ist. C ist so konstruiert, daß jede Transition $t' \in T_E$ genau $i - 1$ Mal vor C auftritt.

Behauptung 2: Für alle $t_1, t_2 \in T_1$ ist die Anzahl der Bedingungen, die kleiner sind als C und mit Stellen aus dem Vorbereich der Transitionen beschriftet sind, gleich, d.h. es gilt:

$$|\{b < C \mid r(b) \in {}^\bullet t_1 \cap {}^\bullet t\}| = |\{b < C \mid r(b) \in {}^\bullet t_2 \cap {}^\bullet t\}|$$

Diese Behauptung folgt direkt aus (iii)(a) und (b) und der Schaltregel für Petrinetze. Wir betrachten nun den Schnitt C_1, der direkt nach C nach dem Schalten von e entsteht. In diesem Schnitt hat der i-te Ablauf von (N_E, t^-) begonnen und wegen (iii)(c) steht keine Transition des Ablaufs in Konflikt mit einer Transition der Umgebung (iii)(c). Folglich wird jede Transition von N_E das i-te Mal auftreten nach C, ohne daß der Ablauf durch die Umgebung von N_E gestört wird.

E_i erfüllt also die Bedingungen (i) und (ii) von Satz 2.2. q.e.d.

2.3 Der Crosstalk-Algorithmus

Zum besseren Verständnis demonstrieren wir nun Transitionsverfeinerung an einem gröseren Beispiel. Wir wenden Transitionsverfeinerung an, um den Crosstalk-Algorithmus [Wal95] zu erklären. Dieser Algorithmus ist ein Kommunikationsprotokoll für zwei Agenten.

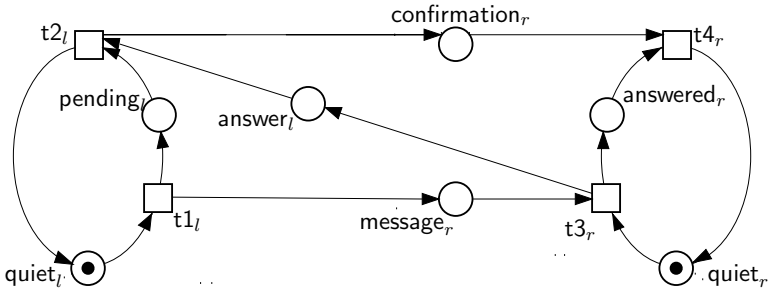


Abb. 2.6: Das Kommunikationsprotokoll Σ_1

Wir betrachten zwei Agenten, die wir den linken und den rechten Agenten nennen. Der linke Agent schickt eine Nachricht an den rechten Agenten. Der rechte Agent beantwortet die Nachricht und schließlich bestätigt der linke Agent die Antwort. Nachdem die Antwort bestätigt ist, beginnt die Kommunikation wieder damit, daß der linke Agent dem rechten einen Nachricht schickt u.s.w.

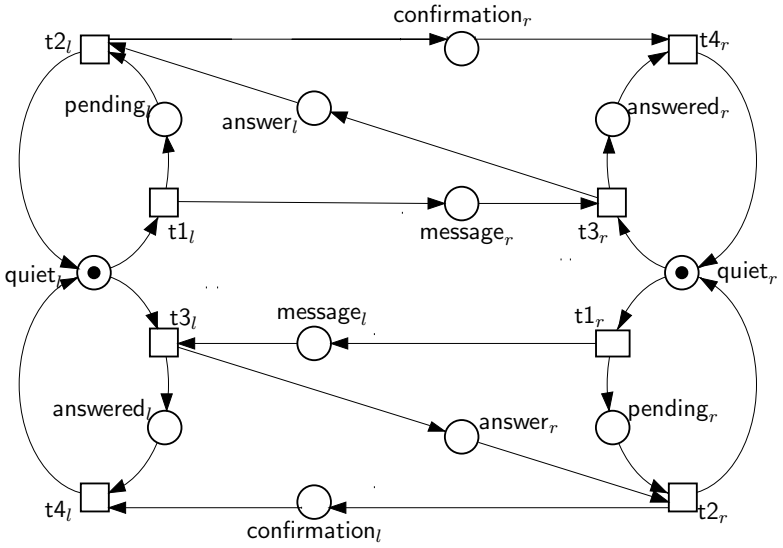


Abb. 2.7: Das symmetrische Protokoll Σ_2

Dieses System ist durch das Petrinetz Σ_1 in Abb. 2.6 modelliert. Der linke Agent ist immer in einem der Zustände quiet und pending, der rechte Agent ist immer in einem der Zustände quiet and answered. Zu Beginn sind beide Agenten quiet. Der linke Agent sendet eine Nachricht ($\mathsf{message_r}$) und wird pending (Transition $\mathsf{t1}_l$). Der rechte Agent empfängt die Nachricht, sendet eine Antwort ($\mathsf{answer_l}$) und geht in den Zustand answered über (Transition $\mathsf{t3}_r$). Der linke Agent im Zustand pending bekommt

die Antwort, sendet eine Bestätigung ($\mathsf{confirmation}_r$) und wird quiet (Transition $\mathsf{t2}_l$). Schließlich erhält der rechte Agent im Zustand answered die Bestätigung und wird wieder quiet (Transition $\mathsf{t4}_r$). Dieses System ist *nachrichtenbasiert*, d.h. Agenten kommunizieren miteinander, indem sie sich gegenseitig Nachrichten schicken. Die Agenten haben keine gemeinsamen Variablen und keine synchronen Aktionen.

Wir betrachten nun ein System, in dem jeder der Agenten sich entschließen kann, eine Nachricht zu senden. Wie im ersten System, soll jede Nachricht beantwortet und jede Antwort bestätigt werden. Das Petrinetz Σ_2 in Abb. 2.7 ist ein erster Versuch, ein solches System zu modellieren. Das System arbeitet wie gewünscht, wenn sich immer nur ein Agent entscheidet, eine Nachricht zu senden und der andere Agent antwortet. Wenn beide Agenten nebenläufig zueinander eine Nachricht abschicken (beide Agenten schalten ihre Transition t1), dann blockiert das System: beide Agenten warten auf eine Antwort, die nie kommt. Diese Situation nennen wir Crosstalk-Situation.

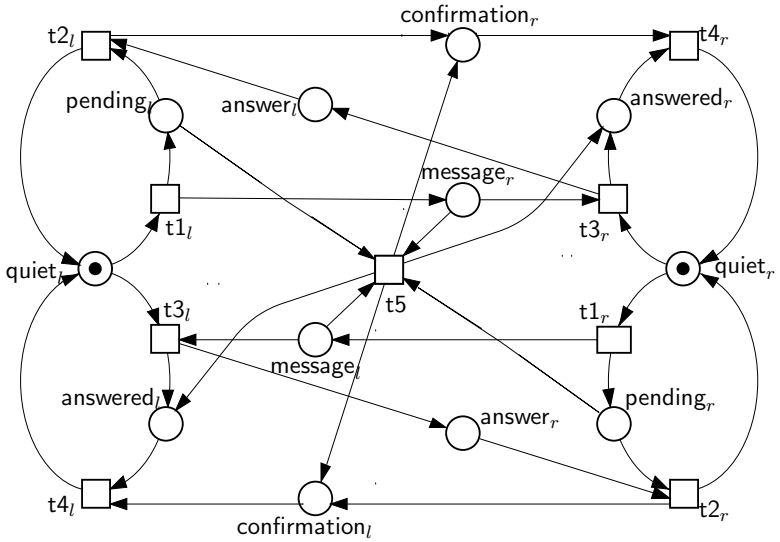


Abb. 2.8: Das System Σ_3

Betrachten wir nun das System Σ_3 , das in Abb. 2.8 modelliert ist. Wir führen hier eine neue Transition t5 ein, die von beiden Agenten gemeinsam ausgeführt wird, wenn beide Agenten eine Nachricht abgeschickt haben und auf Antwort warten. Wir nennen diese Transition die Crosstalk-Transition. Mit der Crosstalk-Transition modellieren wir, daß beide Agenten gemeinsam in der Lage sind, eine Crosstalk-Situation zu erkennen und sie aufzulösen indem sie in den Zustand answered übergehen und sich gegenseitig Bestätigungen schicken. Dieses System blockiert nie. Im Gegensatz zu den ersten beiden Systemen, ist Σ_3 nicht nachrichtenbasiert. Die Transition t5 ist eine synchrone Aktion beider Agenten.

Wir möchten die Crosstalk-Transition so verfeinern, daß das neue System nachrichtenbasiert ist. Deshalb ersetzen wir t5 im System Σ_3 durch zwei Transitionen (siehe in Abb. 2.9). Die Transitionen $\mathsf{t5}_l$ und $\mathsf{t5}_r$ bilden zusammen ein Ersetzungsnetz für die Transition t5.

Das System Σ_4 (dargestellt in Abb. 2.10) ergibt sich wenn wir t5 durch das Erset-

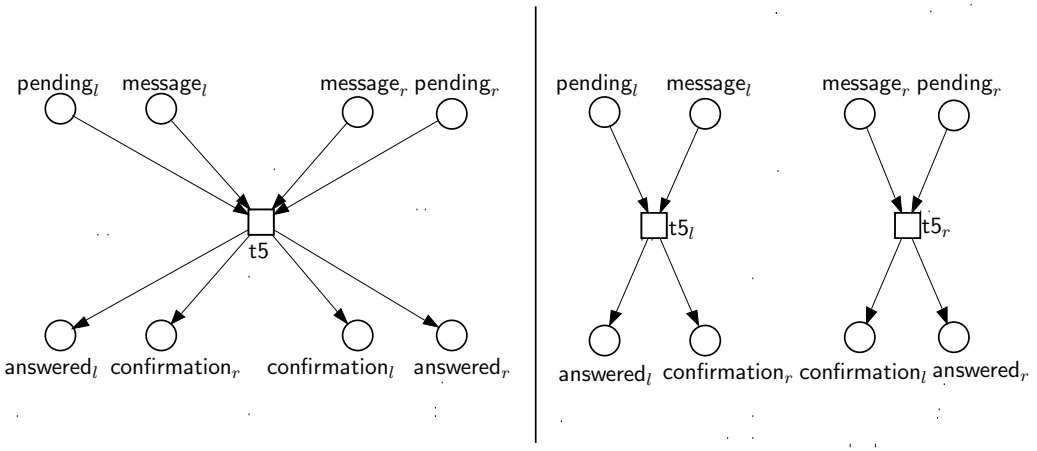


Abb. 2.9: Die Crosstalk-Transition und ein Ersetzungsnetz

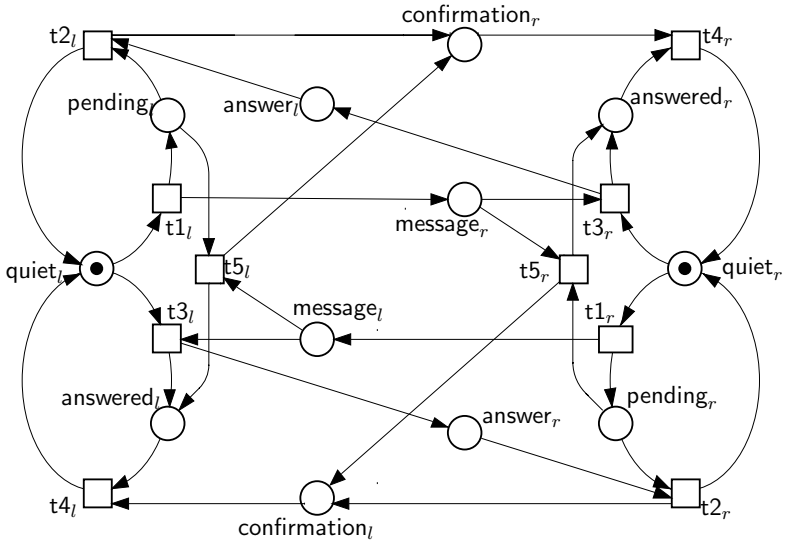


Abb. 2.10: Das System Σ_4: Der Crosstalk-Algorithmus

zungsnetz ersetzen. Jeder Agent kann nun seine Crosstalk-Transition schalten, ohne sich mit dem anderen Agenten zu synchronisieren.

Es ist nicht auf den ersten Blick zu erkennen, ob diese Ersetzung eine Transitionsverfeinerung ist. Kann es sein, daß die linke Crosstalk-Transition schaltet, während die rechte gar nicht aktiviert war? Ist es möglich, daß der linke Agent seine dritte Bestätigung erhält, obwohl er erst zwei Nachrichten abgeschickt hat?

Wir zeigen nun, daß das Ersetzungsnetz aus Abb. 2.9 tatsächlich eine Transitionsverfeinerung ist. Wir benutzen Satz 2.7 zum Beweis. Zuerst zeigen wir durch *Stelleninvaranten*, daß Σ_3 und Σ_4 sicher sind. Eine Stelleninvariante besagt, daß die Summe aller Marken auf einer bestimmten Menge von Stellen in jeder erreichbaren Markierung konstant ist. Zum Beispiel gilt in Σ_3 und Σ_4, daß die Summe der Marken auf den Stellen quiet_l, $\mathsf{pending}_l$ und $\mathsf{answered}_l$ in jeder erreichbaren Markierung

gleich eins ist. Wir bezeichnen dies durch

$$\Box\, \mathsf{quiet}_l + \mathsf{pending}_l + \mathsf{answered}_l = 1 \tag{2.1}$$

Stelleninvarianten können einfach syntaktisch mit der Anfangsmarkierung und der Struktur des Netzes nachgewiesen werden (siehe [Rei98, WWV+97]). Man muss nur zeigen, daß jede Transition beim Schalten genauso viele Marken von Stellen aus der vorgegebenen Menge benötigt, wie sie Marken auf Stellen dieser Menge erzeugt. Durch die obige Invariante wissen wir bereits, daß auf den Stellen quiet_l, $\mathsf{pending}_l$ und $\mathsf{answered}_l$ nie mehr als eine Marke liegt. Um zu zeigen, daß ein Netz sicher ist, brauchen wir zu jeder Stelle eine Stelleninvariante mit der Summe eins, die diese Stelle enthält.

In beiden Systemen, Σ_3 und Σ_4, gelten außerdem die folgenden Stelleninvarianten:

$$\Box\mathsf{quiet}_r + \mathsf{pending}_r + \mathsf{answered}_r = 1 \tag{2.2}$$

$$\Box\mathsf{quiet}_r + \mathsf{message}_l + \mathsf{answer}_r + \mathsf{confirmation}_r + \mathsf{answer}_l = 1 \tag{2.3}$$

$$\Box\mathsf{quiet}_l + \mathsf{message}_r + \mathsf{answer}_l + \mathsf{confirmation}_l + \mathsf{answer}_r = 1 \tag{2.4}$$

Diese Invarianten zeigen bereits, daß Σ_3 und Σ_4 sicher sind.

Die Bedingungen (ii), (iii)(a) und (b) von Satz 2.7 kann man einfach durch einen Blick auf die Netzstruktur überprüfen. Bedingung (iii)(c) gilt, da t5l und t5r in keiner erreichbaren Markierung von Σ_4 in Konflikt zu einer anderen Transition stehen. Um dies zu beweisen benutzen wir ebenfalls Stelleninvarianten.

Sei M ist eine erreichbare Markierung von Σ_4 in der t5l aktiviert ist. Wir untersuchen nun alle Transitionen, die potentiell in Konflikt mit t5l stehen könnten. Transition t2l ist nicht aktiviert, da eine Marke auf $\mathsf{message}_l$ liegt und wegen der Stelleninvariante (2.3) die Stelle answer_l nicht gleichzeitig mit $\mathsf{message}_l$ markiert sein kann. Transition t3l ist nicht aktiviert, da auf der Stelle $\mathsf{pending}_l$ eine Marke liegt und wegen der Stelleninvariante (2.1) die Stelle quiet_l nicht gleichzeitig mit $\mathsf{pending}_l$ markiert sein kann. Die Vorbereiche aller anderen Transitionen sind disjunkt zum Vorbereich von t5l. Folglich steht t5l nie zu einer anderen Transition in Konflikt. Analoge Argumente gelten für t5r. Wir haben bewiesen, daß alle Bedingungen aus Satz 2.7 erfüllt sind.

3 Simultane Transitionsverfeinerung

3.1 Vorschau auf algebraische Petrinetze

Wir führen algebraische Petrinetze erst im zweiten Teil der Arbeit formal ein. In diesem Abschnitt motivieren wir die Einführung simultaner (elementarer) Transitionsverfeinerung mit einer Vorschau auf Transitionsverfeinerung in algebraischen Petrinetzen.

In algebraischen Petrinetzen sind die Marken nicht einfach schwarz, sondern Elemente einer Algebra und damit unterscheidbar. In einem algebraischen Petrinetz kann einen Transition in verschiedenen Modi schalten. Jeder Modus einer Transition in einem algebraischen Petrinetz entspricht einer ganzen Transition in einem elementaren Petrinetz.

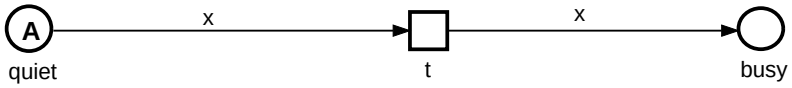


Abb. 3.1: Σ_h: Ein algebraisches Petrinetz mit $A = \{a, b, c\}$

In Abb. 3.1 haben wir ein System Σ_h mit drei Agenten modelliert, die nebenläufig zueinander vom Zustand quiet in den Zustand busy übergehen. A ist eine Menge von Agenten und x ist eine Variable für Agenten. In unserem Beispiel definieren wir $A = \{a, b, c\}$. Transition t schaltet nebenläufig zu sich selbst in verschiedenen Modi, d.h. die Transition schaltet nebenläufig für die Agenten a, b and c. Jedes algebraische System kann entfaltet werden zu einem elementaren System. Die Entfaltung Σ_l von Σ_h ist in Abb. 3.1 dargestellt.

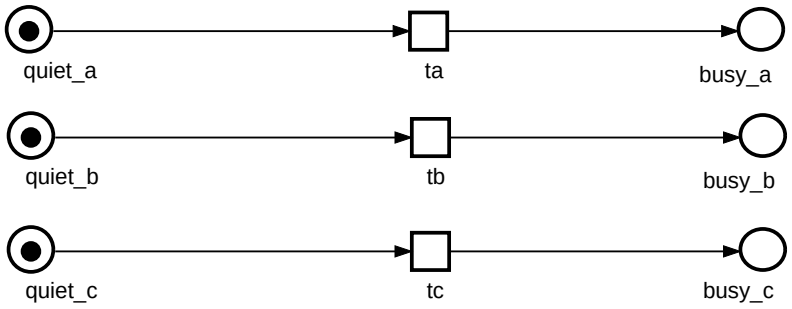


Abb. 3.2: Das System Σ_l

Die Transition t von Σ_h zu verfeinern entspricht einer simultanen Verfeinerung von ta, tb und tc im System Σ_l. Die Verfeinerung einer Transition in einem algebraischen

System ist der simultanen Verfeinerung mehrerer Transitionen in einem elementaren System ähnlich.

3.2 Simultane Transitionsverfeinerung

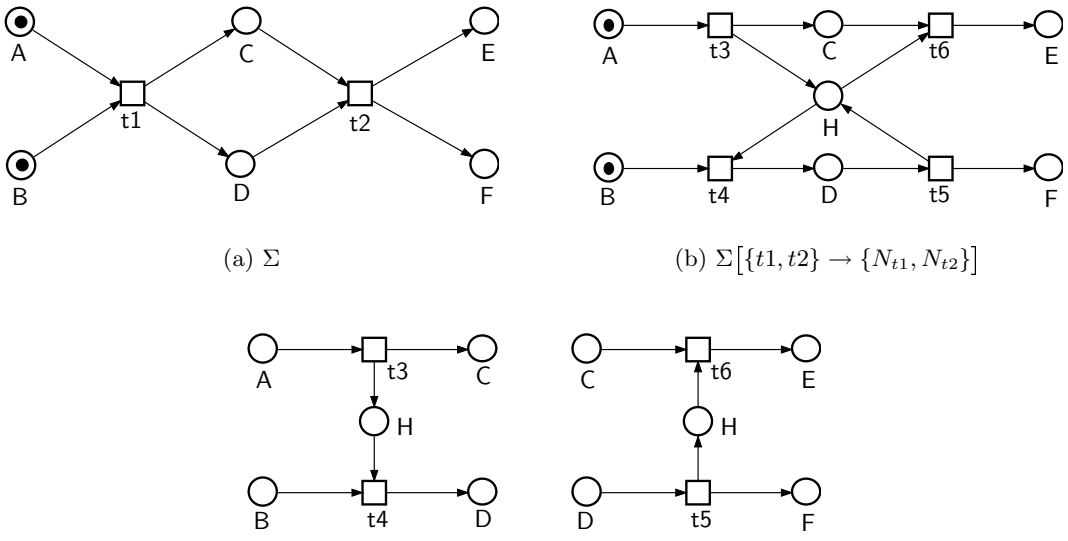
In diesem Abschnitt definieren wir die simultane Verfeinerung einer beliebigen Menge von Transitionen. Simultane Transitionsverfeinerung ist die Grundlage für Transitionsverfeinerung in algebraischen Petrinetzen.

Zuerst definieren wir, wie wir rein syntaktisch eine Menge von Transitionen in einem System ersetzen. Wir nennen $\mathcal{N}$ eine Menge von Ersetzungsnetzen für die Menge von Transitionen T', wenn $\mathcal{N}_E = \{N_t \mid t \in T'\}$ und für jedes $t \in T$, das Netz $N_t = (P_t, T_t, F_t)$ ein Ersetzungsnetz für t ist.

Definition 3.1 (Erweitertes Netz)

Sei $N = (P, T, F)$ ein Netz, sei $T' \subseteq T$ eine Menge von Transitionen des Netzes und sei $\mathcal{N}_E = \{N_t \mid t \in T'\}$ eine Menge von Ersetzungsnetzen für T'. Das *erweiterte Netz* $N\big[T' \to \mathcal{N}_E\big]$ ist das Netz $(\hat{P}, \hat{T}, \hat{F})$, das gegeben ist durch $\hat{P} = P \cup \bigcup_{t \in T'} P_t$, $\hat{T} = (T \setminus T') \cup \bigcup_{t \in T'} T_t$ und $\hat{F} = (F \cup \bigcup_{t \in T'} F_t) \cap (\hat{P} \times \hat{T} \cup \hat{T} \times \hat{P})$.
Für ein System $\Sigma = (N, M_0)$ bezeichnen wir mit $\Sigma\big[T' \to \mathcal{N}_E\big]$ das erweiterte System $(N\big[T' \to \mathcal{N}_E\big], M_0')$ mit $M_0'(p) = M_0(p)$ für $p \in P$ und $M_0'(p) = 0$ sonst. ⋆

Die Ersetzungsnetze für die Transitionen müssen nicht disjunkt sein, d.h. sie können gemeinsame Stellen oder Transitionen enthalten, wie im Beispiel in Abb. 3.3. Beide Ersetzungsnetze enthalten die Stelle H.



(a) Σ

(b) $\Sigma\big[\{t1, t2\} \to \{N_{t1}, N_{t2}\}\big]$

(c) N_{t1} und N_{t2}

Abb. 3.3: Ein System, das erweiterte System und die Ersetzungsnetze

Nun definieren wir ebenfalls kanonisch, wie ein Ablauf durch Abläufe mehrerer Ersetzungsnetze expandiert wird.

Definition 3.2 (Simultane Expansion eines Ablaufs)

Seien Σ ein System, $T' \subseteq T$ eine Menge von Transitionen aus Σ und $\mathcal{N}_E = \{N_t \mid t \in T'\}$ eine Menge von Ersetzungsnetzen für die Transitionen aus T'. Sei außerdem ρ ein Ablauf von Σ. Wir nennen $\rho' = ((B', E', \lessdot'), r')$ eine $\big[T' \to \mathcal{N}_E\big]$-*Expansion* von ρ, wenn wir ρ' aus ρ erhalten, indem wir für jedes $t \in T'$ jedes Auftreten von t durch einen Ablauf von (N_t, t^-) ersetzen (wie in Definition 1.13 beschrieben). ⋆

Definition 3.3 (Simultane Transitionsverfeinerung)

Sei Σ ein System und sei $T' \subseteq T$ eine Menge von Transitionen von Σ. Die Menge $\mathcal{N}_E = \{N_t \mid t \in T'\}$ ist eine *simultane Transitionsverfeinerung* der Transitionen aus T' im System Σ, wenn für jedes $t \in T'$ das Netz N_t ein Ersetzungsnetz für t ist und die Menge aller Abläufe von $\Sigma\big[T' \to \mathcal{N}_E\big]$ gleich der Menge aller $\big[T' \to \mathcal{N}_E\big]$-Expansionen von Abläufen von Σ ist. ⋆

In unserem Beispiel aus Abb. 3.3 ist N_{t1} eine Transitionsverfeinerung für $t1$ und N_{t2} ist eine Transitionsverfeinerung für $t2$. Trotzdem ist $\{N_{t1}, N_{t2}\}$ keine simultane Transitionsverfeinerung, denn der Ablauf des erweiterten Systems, im dem nur $t3$ und $t6$ direkt hintereinander schalten, ist kein expandierter Ablauf von Σ.

3.3 Beweiskriterien für simultane Transitionsverfeinerung

Ein semantisches Kriterium

Zuerst übertragen wir Lemma 2.1 auf simultane Transitionsverfeinerung. Der Beweis ist analog.

Lemma 3.4 Wenn $\mathcal{N}_E$ eine Menge von Ersetzungsnetzen für eine Menge T' von Transitionen aus Σ ist und jeder Ablauf von $\Sigma\big[T' \to \mathcal{N}_E\big]$ eine $\big[T' \to \mathcal{N}_E\big]$-Expansion eines Ablaufs von Σ ist, dann gilt auch die umgekehrte Inklusion: jede $\big[T' \to \mathcal{N}_E\big]$-Expansion eines Ablaufs von Σ ist ein Ablauf von $\Sigma\big[T' \to \mathcal{N}_E\big]$. ⋆

Auch Satz 2.2 läßt sich analog zur elementaren Transitionsverfeinerung für simultane Transitionsverfeinerung formulieren und beweisen.

Satz 3.5 Sei $\mathcal{N}_E$ eine Menge von Ersetzungsnetzen für die Menge T' von Transitionen aus Σ. $\mathcal{N}_E$ ist genau dann eine simultane Transitionsverfeinerung, wenn zu jedem Ablauf $\rho' = ((B', E', \lessdot'), r')$ von $\Sigma\big[T' \to \mathcal{N}_E\big]$ eine Partition aller $\mathcal{N}_E$-Ereignisse des Ablaufs existiert, so daß für jede Äquivalenzklasse A der Partition gilt:

(i) Es gibt zwei Schnitte C_1, C_2 in ρ', so daß

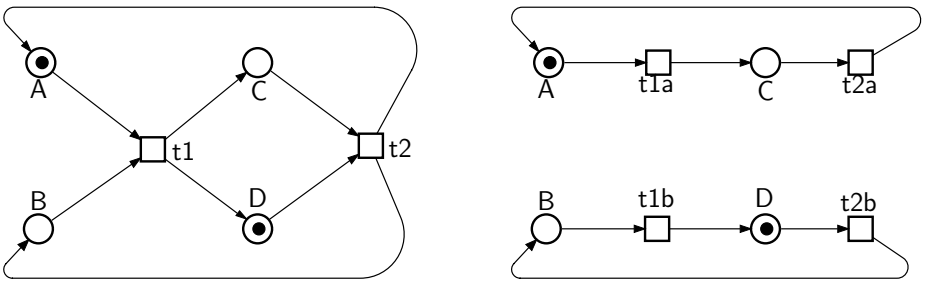
  (a) die Ereignisse zwischen diesen Schnitten genau die Ereignisse der Äquivalenzklasse A sind, d.h. $C_1 < C_2$ und für alle $e \in E'$ gilt

$$C_1 < e < C_2 \qquad \Longleftrightarrow \qquad e \in A.$$

  (b) Für jede Äquivalenzklasse $B \neq A$ gilt: entweder $B < C_1$ oder $B > C_2$. Das bedeutet, daß die Schnitte C_1, C_2 keine anderen Äquivalenzklassen zerschneiden.

(ii) Es gibt ein $t \in T'$, so daß das Teilnetz, das durch die Ereignisse der Äquivalenzklasse induziert wird, zusammen mit der Restriktion von r' auf diese Ereignisse ein Ablauf von (N_t, t^-) ist. ⋆

Für simultane Transitionsverfeinerung ist es im allgemeinen viel schwieriger, Beweiskriterien anzugeben als für elementare Transitionsverfeinerung. Schon das Analogon von Satz 2.3 gilt hier nicht mehr. Wir können also auch für sichere Systeme die Bedingung (i)(b) aus Satz 3.5 nicht weglassen. Wir betrachten dazu wieder ein Beispiel.



(a) Σ (b) Σ'

Abb. 3.4: Ersetzungsnetz und erweitertes System

Wir ersetzen im System Σ (Abb. 3.3(a)) beide Transitionen simultan. Wir ersetzen jede Transition durch zwei zueinander nebenläufige Transitionen. Das Ersetzungsnetz für $\{t1, t2\}$ ist identisch mit dem erweiterten Netz, das dem erweiterten System Σ' zugrunde liegt (Abb. 3.3(b)). Diese Ersetzung ist natürlich keine Verfeinerung, denn in Σ ist keine Transition aktiviert, während in Σ' jede Transition unendlich oft schaltet.

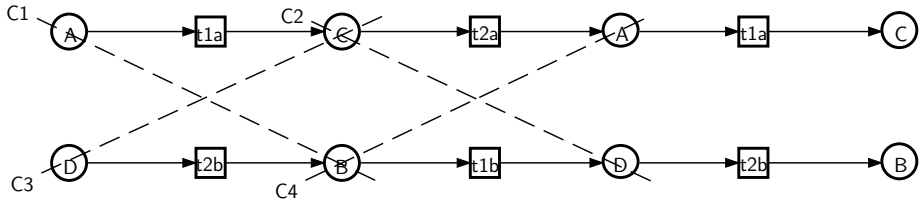


Abb. 3.5: Präfix des verteilten Ablaufs von Σ'

Das System Σ' hat nur einen verteilten Ablauf. In diesem Ablauf kann man die Ereignisse des Ersetzungsnetzes so partitionieren, daß alle Voraussetzungen von Satz 3.5 bis auf (iii) erfüllt sind. Die Äquivalenzklassen durchkreuzen sich gegenseitig (siehe Abb. 3.5). Die Schnitte $C1$ und $C2$ begrenzen eine Äquivalenzklasse, die zu $t1$ gehört. Die Schnitte $C3$ und $C4$ begrenzen eine Äquivalenzklasse, die zu $t2$ gehört. Das System Σ' ist sicher. Im Gegensatz zum elementaren Fall, kann man die Äquivalenzklassen trotzdem nicht so umordnen, daß sie sich nicht überkreuzen.

Auch das einfache Kriterium 2.4 gilt im simultanen Fall nicht mehr ohne Zusatzannahmen: Wir hatten gezeigt, daß jedes (elementare) Ersetzungsnetz eine Verfei-

nerung ist, wenn das Ersetzungsnetz keine verteilte Eingabe erlaubt. Die Sicherheit des Systems garantierte dabei, daß keine Anfangstransition des Ersetzungsnetzes nebenläufig zu sich selbst schalten kann. Damit haben wir ausgeschlossen, daß zwei zueinander nebenläufig begonnene Abläufe des Ersetzungsnetzes durcheinander kommen, also ein Ablauf die Marken benutzt, die im anderen Ablauf erzeugt wurden. Im simultanen Fall reicht es nicht mehr zu fordern, daß das System sicher ist. Wir müssen nun zusätzlich garantieren, daß ein Ablauf des Ersetzungsnetzes, der das Auftreten einer Transition ersetzt, nicht mit einem Ablauf durcheinandergerät, der das Auftreten einer anderen Transition ersetzt.

Beweiskriterien für disjunkte Ersetzungsnetze

Wir zeigen nun, daß die Kriterien für elementare Transitionsverfeinerung jedoch auf simultane Transitionsverfeinerung anwendbar sind, wenn das Ersetzungsnetz für die Menge von Transitionen aus disjunkten Ersetzungsnetzen für jede Transition besteht. Zwei Petrinetze $N_1 = (P_1, T_1, F_1)$ und $N_2 = (P_2, T_2, F_2)$ sind disjunkt, wenn $(P_1 \cup T_1) \cap (P_2 \cup T_2) = \emptyset$.

Simultane Transitionsverfeinerung mit disjunkten Ersetzungsnetzen für die verschiedenen Transitionen sieht auf den ersten Blick nicht sehr interessant aus. Für uns ist es jedoch interessant, denn wir führen simultane Transitionsverfeinerung hauptsächlich ein, um Transitionsverfeinerung in algebraischen Petrinetzen vorzubereiten. In algebraischen Petrinetzen ist es oft so, wie in unserem Beispiel aus Abb. 3.1, daß die Entfaltung einer Transition aus disjunkten Petrinetzen besteht; für jeden Modus ein Petrinetz. Wenn eine solche algebraische Transition ersetzt wird, wird das Ersetzungsnetz auch oft (aber nicht notwendigerweise) aus disjunkten Teilnetzen bestehen.

Satz 3.6 Seien Σ ein System, T' eine Menge von Transitionen aus Σ und $\mathcal{N}_E = \{N_t \mid t \in T'\}$ eine Menge von Ersetzungsnetzen für T'. Wenn für jedes $t \in T'$ das Netz N_t eine Transitionsverfeinerung für die Transition t in Σ ist und die Netze N_t paarweise disjunkt sind, dann ist die Vereinigung $\mathcal{N}_E$ eine simultane Transitionsverfeinerung von T' in Σ. $\star$

Beweis: Dieser Satz folgt direkt aus den Definitionen 1.16 und 3.3. Da die Ersetzungsnetze zweier verschiedener Transitionen disjunkt sind, können sich die Abläufe dieser beiden disjunkten Ersetzungsnetze gegenseitig nicht beeinflussen. q.e.d.

Nicht jede Transition eines algebraischen Petrinetzes wird in disjunkte Teilnetze aufgefaltet, so daß es für jeden Modus genau ein Teilnetz gibt. Meist läßt sich die Menge der Modi einer Transition in Teilmengen zerlegen, so daß das Ersetzungsnetz aus einer Menge von disjunkten Teilnetzen besteht, so daß jedes Teilnetz ein Ersetzungsnetz für eine Teilmenge der Modi ist. In diesem Fall hilft uns die folgende Verallgemeinerung von Satz 3.6:

Satz 3.7 Sei I eine Indexmenge und sei $(T_i)_{i \in I}$ eine Familie von paarweise disjunkten Teilmengen von Transitionen aus Σ. Sei weiter für jedes $i \in I$ die Menge $\mathcal{N}_i$ eine simultane Transitionsverfeinerung von T_i in Σ. Wenn außerdem je zwei Ersetzungsnetze aus verschiedenen $\mathcal{N}_i$ disjunkt sind, dann ist $\bigcup_{i \in I} \mathcal{N}_i$ eine simultane Transitionsverfeinerung von $\bigcup_{i \in I} T_i$ in Σ. $\star$

Beweis: Dieser Satz folgt wie Satz 3.6 direkt aus den Definitionen 1.16 und 3.3.
q.e.d.

Teil II

Modellierung und Verifikation verteilter Algorithmen

In diesem Teil der Arbeit diskutieren wir Modellierung und Verifikation von verteilten Algorithmen, die auf asynchronem Nachrichtenaustausch basieren. Zur Modellierung komplexer verteilter Algorithmen sind einfache Petrinetze oft nicht adäquat. Wir benutzen deshalb die in [Rei91] eingeführten algebraischen Petrinetze. Wir definieren und diskutieren Transitionsverfeinerung für algebraische Petrinetze. Außerdem übertragen wir ein klassisches Verfeinerungskonzept auf algebraische Petrinetze.

4 Modellierung verteilter Algorithmen

4.1 Nachrichtenbasierte Algorithmen

In einem verteilten Algorithmus gibt es, im Gegensatz zu einem nicht-verteilten Algorithmus, mehrere handelnde Einheiten. Eine handelnde Einheit nennen wir einen *Agenten*. Durch einen verteilten Algorithmus wird jedem Agenten eine Handlungsvorschrift zugewiesen.

In dieser Arbeit betrachten wir verteilte Algorithmen, in denen die Agenten ausschließlich über den asynchronen Austausch von Nachrichten kommunizieren (asynchronous message-passing paradigm). Wir nennen diese Algorithmen *nachrichtenbasierte Algorithmen*. Asynchroner Nachrichtenaustauch ist ähnlich wie die Benutzung gemeinsamer Variablen (shared variable paradigm) oder synchroner Nachrichtenaustausch (synchronous message-passing paradigm) eine Annahme an die Architektur, auf der ein Algorithmus ausgeführt wird. In diesem Abschnitt formalisieren wir die Annahmen an die Architektur, auf der ein nachrichtenbasierter Algorithmus ausgeführt wird.

Kommunikationsnetzwerke

Ein *Kommunikationsnetzwerk* (kurz: *Netzwerk*) besteht aus einer Menge von Agenten und einer Menge von bidirektionalen Kommunikationskanälen zwischen verschiedenen Agenten.

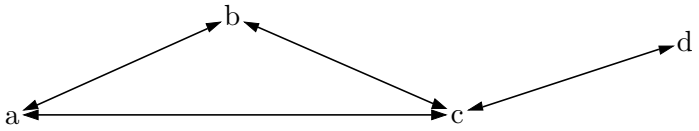


Abb. 4.1: Kommunikationsnetzwerk mit vier Agenten

Zwei Agenten, die durch einen Kommunikationskanal miteinander verbunden sind, heißen *Nachbarn*. Nachbarn können sich gegenseitig Nachrichten schicken. Die Kommunikation findet asynchron statt. Wir gehen davon aus, daß jede Nachricht irgendwann ihren Empfänger erreicht. Jeder Agent kennt seine Nachbarn. Außerdem kann jeder Agent Daten speichern und auf Grundlage dieser Daten Berechnungen ausführen.

Ein *nachrichtenbasierter Algorithmus* auf einem Kommunikationsnetzwerk ist ein verteilter Algorithmus, der jedem Agenten des Netzwerks eine Handlungsvorschrift

zuweist, die er aufgrund lokaler Berechnungen mit den ihm bekannten lokalen Daten und den von seinen Nachbarn empfangenen Nachrichten ausführt.

Mathematisch gesehen ist ein Kommunikationsnetzwerk ein Graph (A, N) mit der Knotenmenge A, den Agenten der Netzwerks, und der Kantenmenge N, den Kommunikationskanälen. Wir repräsentieren einen unidirektionalen Kommunikationskanal vom Agenten a zum Agenten b durch das geordnete Paar (a, b). Einen bidirektionalen Kommunikationskanal repräsentieren wir durch zwei geordnete Paare von Agenten, für jede Richtung ein Paar. Da wir außerdem ausschließen möchten, daß ein Agent zu sich selbst einen Kommunikationskanal hat, ist N immer eine symmetrische, irreflexive Relation in $A \times A$. Wir werden die Begriffe Agent und Knoten bzw. Kommunikationskanal und Kante synonym benutzen.

Die in dieser Arbeit verwendeten graphentheoretischen Begriffe und Resultate sind im Anhang A angegeben. Wir benutzen einen Graph meist als Abstraktion eines Kommunikationsnetzwerks. Wir betrachten deshalb nur endliche Graphen und halten dies (abweichend von klassischen Definitionen) bereits in der Definition fest.

Ein Beispielmodell

Elementare Petrinetze sind gut geeignet zur Modellierung von Protokollen zwischen zwei Agenten. Ein elementares Petrinetzmodell wird aber bei Netzwerken mit mehr als zwei Agenten schnell unübersichtlich, da man jede Aktion jedes einzelnen Agenten darstellen muß. Außerdem wird ein nachrichtenbasierter Algorithmus selten für ein bestimmtes Netzwerk angegeben. Normalerweise kann der Algorithmus auf einer ganzen Klasse von Netzwerken ausgeführt werden, z.B. auf allen zusammenhängenden Netzwerken oder auf allen Ringen. Es ist auch schwer, mit elementaren Petrinetzen Algorithmen zu modellieren, bei denen Daten einen große Rolle spielen.

Wir modellieren solche Algorithmen mit *algebraischen Petrinetzen*. Bevor wir algebraische Petrinetze formal einführen, erklären wir den Formalismus anhand eines Beispiels.

Wir betrachten einen nachrichtenbasierten Algorithmus auf einem Kommunikationsnetzwerk (A, N). Ein konkretes Beispiel für ein Kommunikationsnetzwerk haben wir bereits in Abb. 4.1 angegeben. Dort sind $A = \{a, b, c, d\}$ die Menge der Agenten und $N = \{(a, b), (b, a), (a, c), (c, a), (b, c), (c, b), (c, d), (d, c)\}$ die Menge der Kommunikationskanäle.

Unser Beispielalgorithmus ist eine einfache verteilte Berechnung. Wir abstrahieren in diesem Beispiel davon, was eigentlich berechnet wird. Wir modellieren den Kontrollfluß der Berechnung. Jeder Agent ist immer entweder im Zustand *aktiv* oder im Zustand *passiv*. Ein aktiver Agent kann Nachrichten an seine Nachbarn schicken und er kann spontan in den passiven Zustand übergehen. Ein passiver Agent wird durch den Empfang einer Nachricht aktiv. Wenn alle Agenten passiv sind und keine Nachrichten mehr unterwegs sind, kann keine Aktion mehr ausgeführt werden. Ein solcher Zustand muß nicht erreicht werden. Es gibt endliche und unendliche Abläufe des Algorithmus.

In Abb. 4.1 ist dieser Algorithmus als algebraisches Petrinetz dargestellt. In einem algebraischen Petrinetz sind die Marken Elemente einer Algebra. Jeder Stelle wird

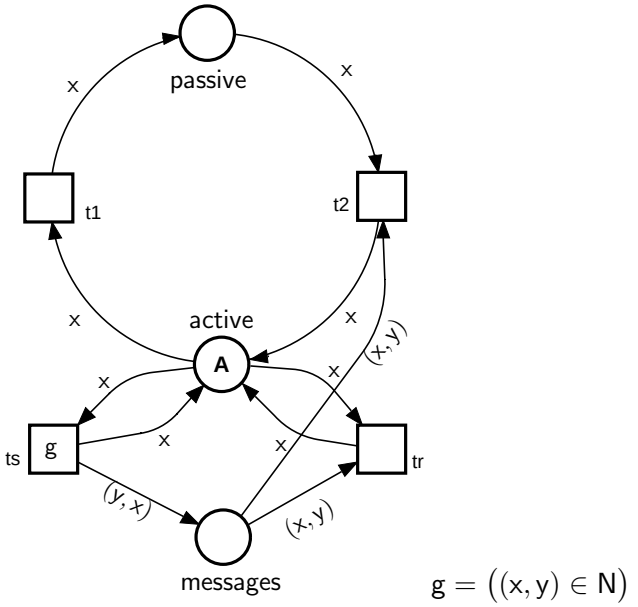


Abb. 4.2: Das Petrinetzmodell Σ_{vb}: Eine verteilte Berechnung

ein Wertebreich (eine Sorte der Algebra) zugeordnet. In unserem Beispiel haben die Stellen active und passive die Sorte A (Menge der Agenten), d.h. jede Marke auf diesen Stellen ist ein Agent des Netzwerks. Eine Marke a auf der Stelle active bedeutet, daß Agent a aktiv ist. Zu Beginn des Algorithmus sind alle Agenten aktiv, d.h. auf der Stelle active liegt die gesamte Agentenmenge A.

Die Stelle messages modelliert die Nachrichtenkanäle. Sie hat die Sorte $A \times A$ (geordnete Paare von Agenten). Eine Marke (a, b) auf der Stelle messages bedeutet, daß eine Nachricht vom Agenten a an den Agenten b unterwegs ist. Die Kanten des Petrinetzes sind mit Termen der Algebra beschriftet. In unserem Beispiel sind x und y Variablen für Agenten. Ob und wie eine Transition schalten kann, hängt davon ab, welche Werte den Variablen zugewiesen werden. Eine konkrete Belegung der Variablen nennen wir einen *Modus* einer Transition.

Die Transition t_1 kann z.B. im Modus $[x = a]$ schalten, wenn einen Marke a auf der Stelle active liegt. Beim Schalten wird diese Marke von der Stelle active entfernt und die Marke a auf die Stelle passive gelegt. Damit ist modelliert, daß ein aktiver Agent passiv wird. Transition t_2 modelliert den Übergang eines passiven Agenten in den aktiven Zustand. Die Transition kann im Modus $[x = a, y = b]$ schalten, wenn a passiv ist und eine Nachricht vom Agenten b an a unterwegs ist. Die Nachricht wird empfangen (und verbraucht), wenn die Transition in diesem Modus schaltet. Ein aktiver Agent kann Nachrichten an seine Nachbarn versenden. Das ist durch die Transition ts modelliert. ts hat eine zusätzliche Aktivierungsbedingung $\big((x, y) \in N\big)$, die garantiert, daß ein Agent nicht an beliebige Agenten des Netzwerks, sondern nur an seine Nachbarn Nachrichten versenden kann. Schließlich wird durch Transition tr modelliert, daß ein aktiver Agent eine Nachricht empfängt.

Wenn alle Agenten passiv sind und keine Nachrichten mehr unterwegs sind, wird der Zustand des Systems nicht mehr verändert. Ein solcher Zustand wird aber nicht in jedem Ablauf erreicht. Es gibt unendliche Abläufe des Systems.

Nachrichten und Agentenzustände

Allgemein unterscheiden wir in einem Petrinetzmodell eines verteilten Algorithmus Stellen, die Zustände von Agenten modellieren und Stellen, die Nachrichtenkanäle modellieren. Wir gehen immer davon aus, daß je zwei verschiedene am Algorithmus beteiligte Agenten verschiedene Namen haben. Einen Agenten a, der lokal dynamische Daten $d_1, \ldots, d_n$ gespeichert hat, modellieren wir als Tupel $(a, d_1, \ldots, d_n)$ (In unserem Beispiel speichern die Agenten keine lokalen dynamischen Daten). Eine Nachricht von Agent b an Agent a modellieren wir als geordnetes Paar (a, b). Wenn die Nachricht für den Algorithmus wichtige Daten $d_1, \ldots, d_n$ enthält, modellieren wir die Nachricht als $(a, b, d_1, \ldots, d_n)$, so daß immer noch der Empfänger an erster und der Absender an zweiter Stelle steht.

Nicht jedes algebraische Petrinetz modelliert einen nachrichtenbasierten Algorithmus. Bei der vorgeschlagenen Modellierung gibt es ein einfaches, hinreichendes Kriterium dafür, daß ein Algorithmus nachrichtenbasiert ist [Des97]: Alle eingehenden Kanten einer Transition sind mit Tupeln (oder Multimengen von Tupeln) beschriftet, deren erste Komponente die gleiche Variable x ist. Dabei ist x immer der handelnde Agent. Dadurch wird garantiert, daß jede Aktion lokal bei einem Agenten ausgeführt wird und nicht mehrere Agenten an einer Aktion beteiligt sind.

Bei datenintensiven Algorithmen, d.h. Algorithmen bei denen jeder Agent lokal viele Daten speichern muß, ist es meist sinnvoll, die Zustände aller Agenten auf einer einzigen Stelle zu repräsentieren. Im Petrinetzmodell ist dann nur noch der Datenfluß graphisch dargestellt. Viele der auf diese Art modellierten Systeme sind sicher, d.h. es gibt nie zwei identische Marken auf einer Stelle. Da jeder Agent nur einmal vorkommt, gibt es für jeden Agenten höchstens eine Marke. Falls es vorkommen kann, daß ein Agent zwei identische Nachrichten von einem Nachbarn erhält, dann kann man das Modell zu einem sicheren Modell umformen, indem wir jeden Agenten mit einem Zähler ausstatten, der die versendeten Nachrichten unterscheidbar macht. Der Zähler zählt die versendeten Nachrichten des Agenten und hängt an jede Nachricht die aktuelle Zahl an. Dadurch kann kein Agent zwei identische Nachrichten verschicken. Diese Umformung kann man so vornehmen, daß alle temporalen Eigenschaften des Systems erhalten bleiben (siehe Abschnitt 5.3).

Weitere Petrinetzmodelle nachrichtenbasierter Algorithmen findet man z.B. in [Rei98, WWV+97]. Beispiele für nachrichtenbasierte Algorithmen findet man in allen Büchern über verteilte Algorithmen, z.B. in [Lyn96, Tel94, Bar96].

4.2 Algebraische Petrinetze

In diesem Abschnitt definieren wir Syntax und Semantik algebraischer Petrinetze. Algebraische Petrinetze wurden in [Rei91] eingeführt. Sie sind verwandt mit gefärbten Petrinetzen aus [Jen92]. Wir benutzen hier die um flexible Kantengewichte erweiterte Form algebraischer Petrinetze [KV98]. Dem mit der Modellierungs- und Verifikationsmethode *Distributed Algorithms' Working Notation (DAWN)* [WWV+97] vertrauten Leser wird in diesem Abschnitt nichts Neues geboten. Die Begriffe und Notationen sind größtenteils übernommen aus [KR96, WWV+97, Rei98].

Algebra

Eine *Algebra* $\mathcal{A} = (U, Op)$ besteht aus einer Menge U (dem Universum der Algebra) und einer Menge Op von Operationen auf U. Jede Operation $f \in Op$ hat eine Funktionalität $f : U_1 \times U_2 \times \ldots U_n \to U_{n+1}$, so daß jedes $U_i \subseteq U$.

Für jede Menge U bezeichnen wir mit 2^U die Potenzmenge, d.h. die Menge aller Teilmengen von U. Mit 2^U_e bezeichnen wir die Menge aller *endlichen* Teilmengen von U. Eine Teilmenge U' eines Universums einer Algebra nennen wir auch *Sorte*. Eine *sortierte Variablenmenge* für eine Algebra $\mathcal{A} = (U, Op)$ ist ein Paar $\mathcal{X} = (X, dom)$ aus einer Menge von Variablen X und einer Funktion $dom : X \to 2^U$, die jeder Variable $x \in X$ eine Sorte $dom(x)$ zuordnet.

Die Menge $\mathcal{T}(\mathcal{A}, \mathcal{X}, U')$ der *Terme* der Sorte U' über der Algebra $\mathcal{A}$ und der sortierten Variablenmenge $\mathcal{X}$ wird induktiv wie folgt definiert:

(i) $x \in \mathcal{T}(\mathcal{A}, \mathcal{X}, U')$ wenn $x \in X$ und $dom(x) \subseteq U'$.

(ii) $f(u_1, \ldots, u_n) \in \mathcal{T}(\mathcal{A}, \mathcal{X}, U')$ wenn $f : U_1 \times U_2 \times \ldots U_n \to U_{n+1}$ und $u_i \in \mathcal{T}(\mathcal{A}, \mathcal{X}, U_i)$ für $i = 1, \ldots, n$ und $U_{n+1} \subseteq U'$.

In dieser Definition wird f als Symbol benutzt, das die Operation f bezeichnet. Zugunsten einer übersichtlichen Notation werden wir auch zukünftig nicht zwischen einer Operation und dem zugehörigen Operationssymbol unterscheiden. Normalerweise geht aus dem Kontext hervor, was gerade gemeint ist. Mit $\mathcal{T}(\mathcal{A}, \mathcal{X})$ bezeichnen wir die Menge aller Terme einer Algebra über $\mathcal{X}$.

Eine *Belegung* für die sortierte Variablenmenge $\mathcal{X} = (X, dom)$ der Algebra $\mathcal{A}$ ist eine Funktion $\beta : X \to U$, die jeder Varablen einen Wert gemäß der Sorte der Variablen zuordnet, d.h. $\beta(x) \in dom(x)$ für alle $x \in X$. Die Auswertung von Termen unter einer Belegung β wird wie üblich induktiv definiert, indem jedem Term ein Wert gemäß der Belegung der im Term vorkommenden Variablen zugeordnet wird. Diese kanonische Erweiterung der Belegung auf die Menge aller Terme bezeichnen wir ebenfalls mit $\beta : \mathcal{T}(\mathcal{A}, \mathcal{X}) \to U$.

Eine *Multimenge* m über einer Menge A ist eine Funktion $m : A \to \mathbb{N}$ die jedem Element $x \in A$ eine natürliche Zahl $m[x]$ zuordnet, die die *Vielfachheit* von x in m angibt. Wir können eine Multimenge auch angeben, indem wir alle Elemente entsprechend ihrer Vielfachheit aufzählen, z. B. $m = [1, 3, 1, 4, 1]$, dann ist $m[1] = 3$. Eine Multimenge m über A ist endlich, wenn $\sum_{x \in A} m[x] \in \mathbb{N}$. Für eine Menge A bezeichnen wir die Menge aller *endlichen* Multimengen über A mit $\mathbb{N}^A_e$. Mit $[\,]$ bezeichnen wir die leere Multimenge, die jedem Element die Zahl Null zuordnet. Die Summe $m_1 + m_2$ zweier Multimengen über A ist elementweise definiert durch $(m_1 + m_2)[x] = m_1[x] + m_2[x]$ für jedes $x \in A$. Wenn $m_1[x] \leq m_2[x]$ für alle $x \in A$, dann schreiben wir $m_1 \leq m_2$. Jede Menge fassen wir auch kanonisch als die Multimenge auf, in der jedes Element der Menge die Vielfachheit 1 hat.

Wir betrachten oft Multimengen, deren Elemente n-Tupel sind. Als Schreibabkürzung für die Projektionsfunktionen benutzen wir die folgende Konvention: $m_{(i)}$ ist die Multimenge, die von allen Elementen aus m nur die i-te Komponente enthält. Wir erweitern dies noch zu: $m_{(i,j)}$ ist die Multimenge, die aus geordneten Paaren besteht und zwar aus der i-ten und j-ten Komponente jedes Elements aus m.

Algebraische Petrinetze

Definition 4.1 (Algebraisches Petrinetz)

Ein *Algebraisches Petrinetz*
$N = (\tilde{N}, \mathcal{A}, \mathcal{X}, d, w, g)$, ist gegeben durch

(i) ein Netz $\tilde{N} = (P, T, F)$,

(ii) eine Algebra $\mathcal{A} = (U, Op)$, die die Sorte $\mathbb{B} = \{\text{true}, \text{false}\}$ enthält,

(iii) eine sortierte Variablenmenge $\mathcal{X} = (X, dom)$ für $\mathcal{A}$,

(iv) eine Funktion $d : P \to 2^U$, die jeder Stelle p eine Sorte $d(p)$ zuordnet, so daß $\mathbb{N}_e^{d(p)} \subseteq U$ (d.h. die Menge der endlichen Multimengen über $d(p)$ ist in U enthalten),

(v) eine Funktion $w : F \to \mathcal{T}(\mathcal{A}, \mathcal{X}, U)$, die jeder Kante f einen Term $w(f)$ zuordnet, so daß gilt: Wenn $f = (p, t) \in P \times T$ oder $f = (t, p) \in T \times P$, dann ist $dom(w(f)) \subseteq 2_e^{d(p)}$ (d.h. die Sorte von $w(f)$ besteht aus den endlichen Teilmengen der Sorte der Stelle p die zu f gehört.) [1]

(vi) eine Funktion $g : T \to \mathcal{T}(\mathcal{A}, \mathcal{X}, \mathbb{B})$, die jeder Transition eine Aktivierungsbedingung (einen Term der Sorte $\{\text{true}, \text{false}\}$) zuordnet. ⋆

Nach dieser Definition kann jedes elementare Petrinetz als ein algebraisches Petrinetz aufgefaßt werden, in dem die Algebra nur die Multimengen über der Menge $\{\bullet\}$ enthält, die Kantengewichtung immer $\{\bullet\}$ ist und alle Aktivierungsbedingungen true sind.

Die Elemente der Sorten der Stellen werden auch *Marken* genannt. Eine *Markierung* des algebraischen Netzes N ist eine Funktion[2] $M : \mathcal{P} \to \mathbb{N}_e^U$, die jeder Stelle $p \in P$ eine endliche Multimenge von Marken der Sorte $d(p)$ zuordnet und allen Stellen $\mathcal{P} \setminus P$ die leere Multimenge zuordnet. Durch eine Markierung wird ein globaler Zustand von N beschrieben. Ein algebraisches Petrinetz N zusammen mit einer Markierung M_0 nennen wir ein *algebraisches System* $\Sigma = (N, M_0)$ (oder kurz: System). Wir nennen M_0 die *Anfangsmarkierung* von Σ.

Eine Markierung wird verändert, wenn eine Transition schaltet. Die Veränderung hängt von der Transition t und der Belegung β der Variablen aus $\mathcal{X}$ ab. Wir nennen β einen *Schaltmodus* (kurz: Modus) der Transition. Eine Transition zusammen mit einem Modus (t, β) nennen wir eine *Aktion*. Wie die Aktion die Markierung verändert, hängt nicht von der gesamten Belegung ab, sondern nur von der Belegung der Variablen, die an ein- oder ausgehenden Kanten oder in der Aktivierungsbedingung

[1] Im Gegensatz zu [KV98, WWV+97], aber ähnlich wie [Rei98] benutzen wir nur Mengenterme und keine Multimengenterme als Kanteninschriften. Diese Einschränkung ist theoretisch nicht notwendig, sie dient allein der angenehmeren Lesbarkeit dieser Arbeit: Um elementare Transitionsverfeinerung so einfach wie möglich zu erklären, haben wir Petrinetze ohne Kantengewichtung benutzt. Diese sind ausreichend zur Entfaltung algebraischer Netze mit Mengentermbeschriftung. Zur Entfaltung algebraischer Netze mit Multimengentermbeschriftung benötigt man elementare Netze mit Kantengewichtung (vgl. dazu Abschnitt auf Seite 63).

[2] Zur Erinnerung: Alle Stellen eines Netzes sind Elemente einer festgelegten universellen Stellenmenge $\mathcal{P}$ (siehe Seite 23).

der Transition vorkommen. Wir unterscheiden deshalb nicht zwischen zwei Aktionen (t, β_1) und (t, β_2), wenn β_1 und β_2 für alle Variablen identisch sind, die an ein- oder ausgehenden Kanten oder in der Aktivierungsbedingung von t vorkommen. Für eine mit (t, β) bezeichnete Aktion ist β nur ein Repräsentant einer Klasse von Belegungen, die sich in den für t wichtigen Variablen nicht unterscheiden.

Zu jeder Aktion (t, β) definieren wir (ähnlich wie im elementaren Fall) zwei Markierungen $(t, \beta)^-$ und $(t, \beta)^+$, die gerade die von der Aktion benötigten Marken bzw. die von der Aktion erzeugten Marken beschreiben:

$$\begin{aligned}(t, \beta)^-(p) &= \beta(w(p, t)) \\ (t, \beta)^+(p) &= \beta(w(t, p)) \qquad \text{für alle } p \in \mathcal{P}\end{aligned}$$

Eine Aktion (t, β) ist *aktiviert* in der Markierung M, wenn $(t, \beta)^- \leq M$ und die Aktivierungsbedingung gilt, also $\beta(g(t)) = \text{true}$ ist. Eine Transition t ist *aktiviert* in der Markierung M, wenn es einen Modus β gibt, so daß die Aktion (t, β) in der Markierung M aktiviert ist. Eine aktivierte Aktion (bzw. Transition) kann schalten. Das *Schalten* der Aktion (t, β) (bzw. der Transition t im Modus β) führt zu einer Markierung M', die gegeben ist durch:

$$M' + (t, \beta)^- = M + (t, \beta)^+$$

Dieses Schalten nennen wir einen *Schritt* und bezeichnen es mit $M \overset{(t,\beta)}{\rightarrow} M'$.

Zwei Aktionen (t_1, β_1) und (t_2, β_2) stehen in *Konflikt* in einer Markierung M, wenn sie verschieden[3] sind und beide Aktionen in M aktiviert sind, aber ihre Vorbereiche nicht disjunkt sind, d.h. es gibt eine Stelle p und ein Element a der Sorte von p, so $(t_1, \beta_1)^-(p)[a] > 0$ und $(t_2, \beta_2)^-(p)[a] > 0$.

Eine Markierung M' ist von der Markierung M aus *erreichbar*, wenn es eine (möglicherweise leere) Folge von Schritten $M_0 \overset{(t_1,\beta_1)}{\rightarrow} M_1 \ldots M_{n-1} \overset{(t_n,\beta_n)}{\rightarrow} M_n$ gibt, so daß $M = M_0$ und $M' = M_n$ ist. Eine Markierung ist *im System* Σ *erreichbar*, wenn sie von der Anfangsmarkierung des Systems aus erreichbar ist. Ein Markierung eines Systems heißt *tot*, wenn sie erreichbar ist und keine Aktion des Systems aktiviert ist.

Eine Markierung M ist *sicher*, wenn es keine Stelle gibt, auf der zwei identische Marken liegen, d.h. $M(p)[x] \leq 1$ für alle $p \in \mathcal{P}$ und alle $x \in d(p)$. Ein System ist sicher, wenn jede erreichbare Markierung sicher ist. In einem sicheren System können wir die Multimengen von Marken, die jeder Stelle in einer Markierung zugeordnet werden auch als Mengen auffassen.

Sequentielle Abläufe

Da wir in diesem Teil der Arbeit auch Verfeinerungsbegriffe betrachten, denen eine sequentielle Semantik zugrunde liegt, definieren wir auch sequentielle Abläufe für algebraische Petrinetze. Jeder sequentielle Ablauf unterliegt der Progressannahme, daß jede aktivierte Transition irgendwann schaltet oder eine dazu in Konflikt stehende Transition schaltet.

[3] Zur Erinnerung: Zwei Aktionen (t_1, β_1) und (t_2, β_2) sind verschieden, wenn $t_1 \neq t_2$ oder wenn es eine Variable x gibt, die an einer ein- oder ausgehenden Kante oder in der Aktivierungsbedingung von t_1 vorkommt, so daß $\beta_1(x) \neq \beta_2(x)$.

Definition 4.2 (Sequentieller Ablauf)

Ein *sequentieller Ablauf* eines Systems $\Sigma = (N, M)$ ist eine Folge $M_0 \overset{(t_1,\beta_1)}{\rightarrow} M_1 \overset{(t_2,\beta_2)}{\rightarrow} M_2, \ldots$ von Schritten von Σ, so daß gilt:

(i) $M_0 = M$ und

(ii) (Progressannahme) wenn die Aktion (t, β) in M_i aktiviert ist, dann gibt es eine Markierung M_j mit $i < j$, so daß $M_{j-1} \overset{(t,\beta)}{\rightarrow} M_j$ ein Schritt der Folge ist oder es gibt eine mit (t, β) in M_{j-1} in Konflikt stehende Aktion (t', β'), so daß $M_{j-1} \overset{(t',\beta')}{\rightarrow} M_j$ ein Schritt der Folge ist. ⋆

Verteilte Abläufe

Wir definieren nun *verteilte Abläufe* als Halbordnungssemantik für algebraische Petrinetze. Wie im elementaren Fall benutzen wir beschriftete Kausalnetze zur Darstellung verteilter Abläufe. Wir beschriften die Ereignisse des Kausalnetzes dabei mit Aktionen des Systems und die Bedingungen mit Marken des Systems (also mit einem geordneten Paar, bestehend aus einer Stelle des Systems und einem Element der Stellensorte).

Definition 4.3 (Σ-Beschriftung)

Seien Σ ein algebraisches System und $K = (B, E, \lessdot)$ ein Kausalnetz. Eine Σ-*Beschriftung* von K ist eine Abbildung r, die jedem Ereignis $e \in E$ eine Aktion $r(e) = (t, \beta)$ aus Σ zuordnet und jeder Bedingung $b \in B$ ein Paar $r(b) = (p, a)$, so daß p eine Stelle aus Σ ist und $a \in d(p)$ ein Element aus der Sorte von p ist. ⋆

Durch eine Σ-Beschriftung kann wieder jeder endlichen co-Menge C von K eine Markierung von Σ zugeordnet werden. Wir bezeichnen diese Markierung mit $r(C)$ und definieren $r(C) : P \rightarrow \mathbb{N}_e^U$ mit $r(C)(p)[a] = |\{b \in C \mid r(b) = (p, a)\}|$.

Definition 4.4 (Prozeß)

Sei $\Sigma = (N, M_0)$ ein algebraisches System. Ein Paar $\rho = (K, r)$ ist ein *Prozeß* von Σ, wenn $K = (B, E, \lessdot)$ ein Kausalnetz und r eine Σ-Beschriftung sind, so daß

(i) $r({}^\circ\mathrm{K}) = M_0$ und

(ii) für jedes Ereignis $e \in E$ und jede Transition (t, β) gilt: Wenn $r(e) = (t, \beta)$, dann gilt

$$r({}^\bullet e) = (t, \beta)^- \quad \text{und} \quad r(e^\bullet) = (t, \beta)^+$$ ⋆

Bedingung (i) sichert, daß der Anfangsschnitt auf die Anfangsmarkierung des algebraischen Petrinetzes abgebildet wird. In Bedingung (ii) ist formalisiert, daß jedes Ereignis e mit dem Schalten einer Transition t des algebraisches Petrinetzes in einem bestimmten Modus assoziiert wird. Ein Präfix eines Prozesses ist analog zum elementaren Fall definiert (siehe Definition 1.8). Wie im elementaren Fall definieren wir einen verteilten Ablauf als Prozeß, der maximal bzgl. der Präfixrelation ist. Dies ist äquivalent zu der Progressannahme, daß am Ende des Ablaufs keine Transition mehr aktiviert ist. In sicheren Systemen ist diese Annahme äquivalent zu der Annahme, daß jede aktivierte Transition schaltet oder eine konfliktäre Transition schaltet.

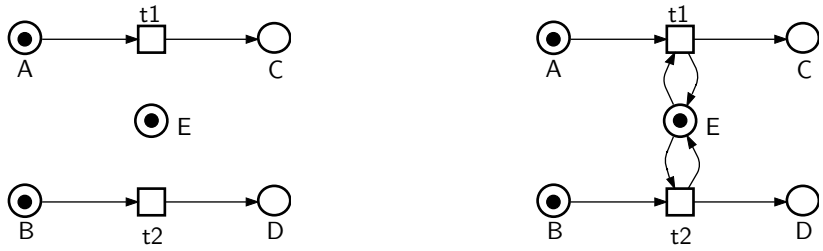
Definition 4.5 (Verteilter Ablauf)

Sei $\Sigma = (N, M_0)$ ein algebraisches System. Ein Paar $\rho = (K, r)$ ist ein *verteilter Ablauf* (kurz: Ablauf) von Σ, wenn ρ ein Prozeß von Σ ist und alle Transitionen $t \in T$ in $r(\mathrm{K}^\circ)$ deaktiviert sind, also für jede Aktion (t, β) und jede endliche co-Menge $C \subseteq \mathrm{K}^\circ$ gilt $(t, \beta)^- \not\leq r(C)$ oder $\beta(g(t)) = false$. Die Menge aller verteilten Abläufe von Σ bezeichnen wir mit $\mathcal{R}(\Sigma)$. ⋆

Bemerkung 4.6

- Eine Markierung M von Σ ist genau dann erreichbar, wenn es einen verteilten Ablauf mit einem Schnitt C gibt, so daß $r(C) = M$.
- In einem sicheren System ist die Menge aller Sequentialisierungen von verteilten Abläufen gleich der Menge aller sequentiellen Abläufe. ⋆

Die zweite Bemerkung gilt, da in einem sicheren System unsere Progreßannahme für sequentielle Abläufe mit der Progreßannahme für verteilte Abläufe zusammenfällt. Das bedeutet, daß zwei sichere Systeme, die die gleichen verteilten Abläufe haben, auch die gleichen sequentiellen Abläufe haben. Umgekehrt kann man jedoch nicht schließen, daß sichere Systeme, die die gleichen sequentiellen Abläufe haben, auch die gleichen verteilten Abläufe haben (siehe dazu auch [CMP87]). Ein Beispiel dafür ist in Abb. 4.3 angegeben.



(a) Σ_1 (b) Σ_2

Abb. 4.3: Gleiche sequentielle, aber ungleiche verteilte Abläufe

Das System Σ_1 hat nur einen verteilten Ablauf, es gibt keine Konflikte. Das System Σ_2 hat zwei verteilte Abläufe, je nachdem, welche Transition zuerst schaltet. Beide Systeme haben aber die gleichen sequentiellen Abläufe. Dies ist auch ein Beispiel dafür, daß in den verteilten Abläufen mehr Information als in den sequentiellen Abläufen eines Systems steckt.

Entfaltung eines algebraischen Petrinetzes

Wir können jedes algebraische System Σ in ein elementares System Σ^e umwandeln, so daß beide Systeme dieselben verteilten Abläufe haben. Wir nennen Σ^e die *Entfaltung* von Σ. Die Menge der Transitionen von Σ^e ist dabei gleich der Menge der Aktionen von Σ und die Menge der Stellen ist gleich der Menge aller möglichen Marken, also

wie in Def. 4.3 der Menge aller Paare (p, a), so daß p eine Stelle aus Σ ist und $a \in d(p)$ ein Element aus der Sorte von p ist. Die Flußrelation von Σ^e hängt von den Kanteninschriften und den Aktivierungsbedingungen von Σ ab.

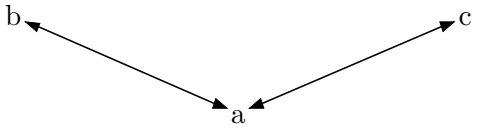


Abb. 4.4: Kommunikationsnetzwerk mit drei Agenten

Wir betrachten noch einmal das algebraische System Σ_{vb} auf Seite 57. Da eine Entfaltung für das Kommunikationsnetzwerk mit vier Agenten bereits ziemlich unübersichtlich ist, betrachten wir das noch einfachere Kommunikationsnetzwerk in Abb. 4.4. Mit einem Kommunikationsnetzwerk ist die Algebra für Σ_{vb} soweit festgelegt, daß wir die Entfaltung bilden können. Die Entfaltung des algebraischen Systems mit dem Netzwerk aus Abb. 4.4 ist in Abb. 4.7 angegeben.

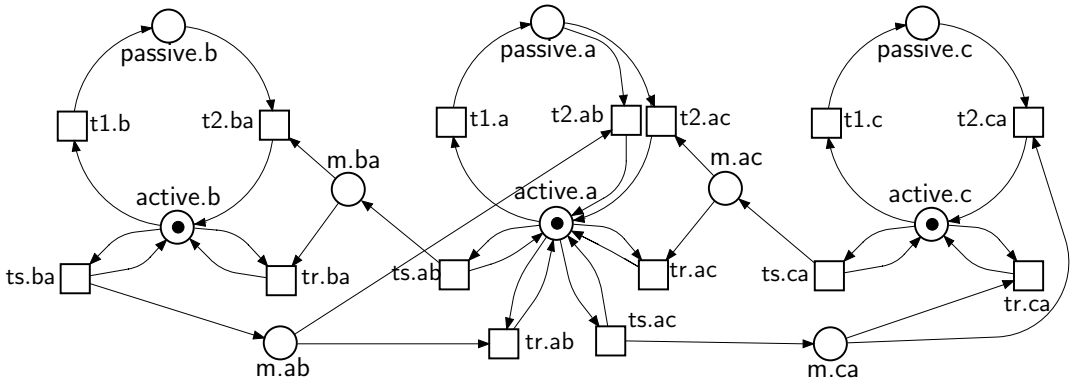


Abb. 4.5: Entfaltung von Σ_{vb} für das Netzwerk aus drei Agenten

Diese Entfaltung haben wir folgendermaßen erhalten: Wir betrachten zu jeder Transition des algebraischen Netzes jede mögliche Belegung von Variablen, die an ein- oder ausgehenden Kanten der Transition oder in der Aktivierungsbedingung vorkommen. Für jede Belegung, in der die Aktivierungsbedingung der Transition erfüllt ist, zeichnen wir eine Transition in der Entfaltung. In unserem Bild haben wir t.ab als Notation für die Transition t im Modus $[x = a, y = b]$ benutzt bzw. t.a als Notation für die Transition t im Modus $[x = a]$, wenn die Belegung von y keine Rolle spielt. Es gibt zum Agenten a zwei Transitionen (ts.ab, ts.ac), die das Abschicken einer Nachricht an einen Nachbarn modellieren, nämlich für jeden Nachbarn eine. Andererseits gibt es keine Transition, die das Abschicken einer Nachricht von b an c modelliert, da b und c keine Nachbarn sind und die Aktivierungsbedingung $(b, c) \in N$ nicht erfüllt ist.

Zu jeder Stelle p des algebraischen Netzes bilden wir für jedes Element z der Sorte von p eine Stelle $p.z$. Wenn die Aktion (t, β) die Marke $p.z$ zum Schalten benötigt (also wenn $(t, \beta)^-(p)[z] = 1$), dann zeichnen wir eine Kante von der Stelle $p.z$ zur elementaren Transition t.β. Wenn umgekehrt die Marke $p.z$ von (t, β) erzeugt wird, dann zeichnen wir eine Kante von der Transition t.β zur Stelle $p.z$.

Schließlich markieren wir die Stellen entsprechend der Anfangsmarkierung des algebraischen Netzes (alle Agenten sind aktiv und keine Nachricht ist unterwegs.) Die Stellen $p.z$, die nach diesem Verfahren anfangs nicht markiert sind *und* keine Transition im Vorbereich haben, zeichnen wir nicht, da sie nie markiert werden. Formal gehören sie jedoch zum entfalteten Netz dazu. In unserem Beispiel gehört das Paar (b, c) zur Sorte der Stelle message, wird aber von keiner Aktion erzeugt oder verbraucht, deshalb kommt die Stelle m.bc im Bild nicht vor.

Definition 4.7 (Entfaltung)

Sei $N = ((P, T, F), \mathcal{A}, \mathcal{X}, d, w, g)$ ein algebraisches Petrinetz. Das elementare Petrinetz $N^e = (P^e, T^e, F^e)$ ist die *Entfaltung* von N, wenn die folgenden Bedingungen gelten:

(i) $P^e = \{(p, z) \mid p \in P \text{ und } z \in d(p)\}$

(ii) $T^e = \{(t, \beta) \mid t \in T, \beta \text{ ist eine Belegung von } \mathcal{X} \text{ mit } \beta(g(t)) = \text{true}\}$[4]

(iii) $((p, a), (t, \beta)) \in F^e$ gdw. $(t, \beta)^-(p)[a] \neq 0$ und
$((t, \beta), (p, a)) \in F^e$ gdw. $(t, \beta)^+(p)[a] \neq 0$

Sei nun $\Sigma = (N, M_0)$ ein algebraisches System mit $N = ((P, T, F), \mathcal{A}, \mathcal{X}, d, w, g)$. Das elementare System $\Sigma^e = (N^e, M_0^e)$ mit $N^e = (P^e, T^e, F^e)$ ist die *Entfaltung* von Σ, wenn N^e die Enfaltung von N ist und wenn für die Anfangsmarkierungen $M_0^e[(p, z)] = M_0(p)[z]$ gilt. ⋆

Satz 4.8 Ein algebraisches System Σ und seine Entfaltung Σ^e haben dieselben Abläufe. ⋆

Beweis: Direkt aus den Definitionen 4.7 und 4.3 folgt, daß jede Σ-Beschriftung eines Kausalnetzes eine Σ^e-Beschriftung ist. Nun überprüft man leicht, daß die Bedingungen aus den Definitionen 4.4 und 4.5 genau dann für Σ erfüllt sind, wenn sie für Σ^e erfüllt sind. q.e.d.

Folgerung 4.9 Ein algebraisches System ist genau dann sicher, wenn seine Entfaltung sicher ist. ⋆

4.3 FIFO-Kanäle

In vielen nachrichtenbasierten Algorithmen spielen FIFO-Kanäle eine Rolle. FIFO (First-In-First-Out) ist eine Annahme an die Architektur, auf der ein Algorithmus arbeitet. Ein Nachrichtenkanal von einem Agenten a zu einem Agenten b ist FIFO, wenn b alle Nachrichten in derselben Reihenfolge empfängt, in der a sie abgeschickt

[4] Auch hier halten wir uns wieder an die Konvention von Seite 61, daß wir zwei Aktionen nicht unterscheiden, wenn sie sich in den Variablen, die an den ein- und ausgehenden Kanten und in den Aktivierungsbedingungen vorkommen, nicht unterscheiden. In unserem Beispiel gibt es für den Übergang eines Agenten von active nach passive eigentlich drei elementare Transitionen mit identischem Vor- und Nachbereich (für jede Belegung von y eine). Wir zeichnen dafür nur eine Transition, denn die Belegung von y hat keinen Einfluß.

hat. Die Nachrichtenstellen in algebraischen Petrinetzen modellieren ungeordnete Nachrichtenkanäle, keine FIFO-Kanäle.

In diesem Abschnitt zeigen wir, wie wir FIFO-Kanäle in algebraischen Petrinetzen mit Hilfe von Zählern implementieren können. Die in diesem Abschnitt definierten FIFO-Stellen haben nichts mit den von Finkel und Memmi in [FM82] definierten FIFO-Netzen zu tun. FIFO-Netze sind ein ausdrucksstärkerer Formalismus als Petrinetze, denn FIFO-Netze sind im Gegensatz zu Petrinetzen turingmächtig [Tix96, Sta83]. Unsere FIFO-Stellen dagegen sind eine reine Schreibabkürzung und erhöhen die Ausdrucksstärke unseres Formalismus nicht.

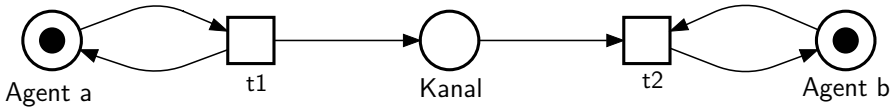


Abb. 4.6: Ein ungeordneter Nachrichtenkanal von a nach b

Zuerst betrachten wir zwei Agenten a und b und einen Nachrichtenkanal von a nach b (Abb. 4.6). Die linke Stelle zeigt den Agenten a und die rechte Stelle den Agenten b. Der Agent a kann eine Nachricht an b schicken (Transition $t1$). Der Inhalt der Nachricht ist hier uninteressant. Der Agent b empfängt die Nachricht (Transition $t2$). Der Nachrichtenkanal ist ungeordnet, d.h. wenn a mehrere Nachrichten abschickt, empfängt b diese nicht unbedingt in derselben Reihenfolge, in der sie abgeschickt wurden.

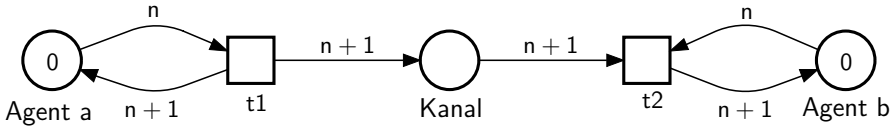


Abb. 4.7: Ein geordneter Nachrichtenkanal von a nach b

Im Modell in Abb. 4.7 zählt a die Nachrichten, die er versendet und b die Nachrichten die er empfängt. Zu Beginn sind die Zähler der Agenten auf null gesetzt. Jeder Agent erhöht seinen Zähler um eins, wenn er eine Nachricht versendet bzw. empfängt. Außerdem bekommt jede Nachricht eine Nummer und b darf eine Nachricht mit der Nummer $i + 1$ nur dann empfangen, wenn er vorher schon i Nachrichten empfangen hat. In diesem Modell bleibt die Reihenfolge der Nachrichten beim Empfangen erhalten. Wir benutzen als Abkürzung für diese Zählerkonstruktion ein besonderes Symbol: einen Doppelkreis.

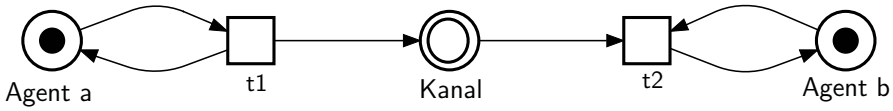


Abb. 4.8: Ein FIFO-Kanal von a nach b

Das Modell in Abb. 4.8 ist also nur eine Schreibabkürzung des Modells in Abb. 4.7. Im Modell in Abb. 4.8 ist garantiert, daß die Nachrichten in derselben Reihenfol-

ge empfangen werden, wie sie versendet werden, obwohl die Zähler nicht explizit modelliert sind.

Wenn wir in einem Kommunikationsnetzwerk FIFO-Kanäle durch Zähler implementieren wollen, dann bekommt jeder Agent viele verschiedene Zähler, nämlich für jeden Nachbarn zwei Zähler, einen zum Zählen der versendeten Nachrichten und einen zum Zählen der empfangenen Nachrichten. Wenn wir eine FIFO-Stelle zur Modellierung eines nachrichtenbasierten Algorithmus benutzen, dann können wir in der Verifikation des Algorithmus den folgenden Satz benutzen. Wir gehen davon aus, das der Algorithmus gemäß der Konvention auf Seite 58 modelliert ist, d.h. jede Nachricht ist ein Tupel dessen erste Position den Empfänger und dessen zweite Position den Empfänger der Nachricht angibt. Mit $(a, b, \dots)$ bezeichnen wir eine Nachricht von b an a mit beliebigem Inhalt.

Satz 4.10 Sei Σ ein algebraisches System und p eine FIFO-Stelle aus Σ. Für jeden Ablauf $((B, E, \lessdot), r)$ und alle Bedingungen $b_1, b_2 \in B$ gilt: Wenn

$$\begin{aligned} &r(b_1) = (p; x, y, \dots) \text{ und} \\ &r(b_2) = (p; x, y, \dots) \text{ und} \\ &{}^\bullet b_1 < {}^\bullet b_2, \qquad \text{dann ist} \quad b_2^\bullet \not< b_1^\bullet. \end{aligned}$$

Für sequentielle Agenten ist $b_2^\bullet \not< b_1^\bullet$ äquivalent zu $b_1^\bullet < b_2^\bullet$. ⋆

4.4 Eigenschaften von verteilten Algorithmen

Zur Beschreibung von Eigenschaften verteilter Algorithmen benutzen wir Formeln einer speziellen temporalen Logik, die auf die vorher definierten algebraischen Systeme abgestimmt ist. Diese Logik ist Teil der *Distributed Algorithms' Working Notation (DAWN)* [Rei98, WWV+97]. Wir geben hier nur die Syntax und Semantik dieser Formeln an. Beweistechniken findet man ebenfalls in [Rei98, WWV+97].

Zustandsaussagen

Mit Zustandsaussagen formalisieren wir Aussagen über Zustände eines algebraischen Netzes. Ein Zustand eines Netzes ist durch eine Markierung und eine Belegung gegeben. Als Grundbausteine einer Zustandsaussage verwenden wir deshalb die Namen der Stellen des Netzes und die booleschen Terme (d.h. die Terme der Sorte $\{\text{true}, \text{false}\}$) der Algebra des Netzes. Jeder Term wird durch die aktuelle Belegung interpretiert und jeder Stellenname durch die aktuelle Belegung der Stelle. Zum Beispiel gilt im System Σ_{vb} (siehe Abb. 4.1) im Anfangszustand die Zustandsaussage $\mathsf{active} = A$. Diese Aussage hängt nur von der Markierung ab. Ob die Zustandsaussage $(x, y) \in N$ gilt, hängt dagegen nur von der Belegung der Variablen x und y ab.

Definition 4.11 (Zustandsaussage)

Sei $N = ((P, T, F), \mathcal{A}, \mathcal{X}, d, w, g)$ ein algebraisches Petrinetz. Wir definieren die Menge $\mathrm{ZA}(N)$ der Zustandsaussagen über N wie folgt:

(i) $\mathcal{T}(\mathcal{A}, \mathcal{X} \cup P, \{\text{true}, \text{false}\}) \subseteq \text{ZA}(N)$, d.h. alle booleschen Terme der Algebra $\mathcal{A}$ über der Variablenmenge $\mathcal{X} \cup P$ sind Zustandsaussagen über N.

(ii) Wenn $\varphi, \varphi' \in \text{ZA}(N)$, dann ist $(\varphi \wedge \varphi') \in \text{ZA}(N)$.

(iii) Wenn $\varphi \in \text{ZA}(N)$, dann ist $\neg\varphi \in \text{ZA}(N)$.

(iv) Wenn $\varphi \in \text{ZA}(N)$ und $x \in \mathcal{X}$, dann ist $(\exists x \varphi) \in \text{ZA}(N)$.

(v) Es gibt keine anderen Zustandsaussagen über N. ⋆

Wie bereits erwähnt, interpretieren wir eine Zustandsaussage in einer Markierung M zusammen mit einer Belegung β der Variablen. Die Auswertung eines booleschen Terms $\varphi \in \mathcal{T}(\mathcal{A}, \mathcal{X} \cup P, \{\text{true}, \text{false}\}) \subseteq \text{ZA}(N)$ unter M und β bezeichnen wir mit $\beta_M(\varphi)$. Dabei werten wir die Variablen aus $\mathcal{X}$ so aus, wie durch β angegeben und die Variablen $p \in P$ werten wir durch $M(p)$ aus.

Definition 4.12 (Gültigkeit einer Zustandsaussage)

Eine Zustandsaussage φ ist genau dann in einer Markierung M und einer Belegung β gültig (in Symbolen: $M, \beta \models \varphi$), wenn

(i) $\varphi \in \mathcal{T}(\mathcal{A}, \mathcal{X} \cup P, \{\text{true}, \text{false}\})$ und $\beta_M(\varphi) = \text{true}$ oder

(ii) $\varphi = (\varphi_1 \wedge \varphi_2)$ und $M, \beta \models \varphi_1$ und $M, \beta \models \varphi_2$ oder

(iii) $\varphi = \neg\varphi_1$ und es gilt nicht $M, \beta \models \varphi_1$ oder

(iv) $\varphi = \exists x \varphi_1$ und es gibt eine Belegung β' mit $\beta(y) = \beta'(y)$ für alle $y \neq x$, so daß $M, \beta' \models \varphi_1$.

Eine Zustandsaussage φ ist gültig in einer Markierung M (in Symbolen: $M \models \varphi$), wenn φ in M unter jeder Belegung β gültig ist. ⋆

Temporale Aussagen

Zustandsaussagen sind die Grundbausteine von temporalen Aussagen. Temporale Aussagen interpretieren wir über den Schnitten von verteilten Abläufen. Wir definieren zunächst nur den temporalen Operator $\Box$ und bauen alle temporalen Aussagen induktiv aus $\Box, \wedge, \neg$ und den booleschen Termen der Algebra auf. Wir führen dann andere temporale Operatoren als Abkürzungen von Kombinationen aus $\Box, \wedge$ und $\neg$ ein. Die Gültigkeit einer temporalen Aussage wird wieder induktiv über den Aufbau der Formeln definiert.

Definition 4.13 (Gültigkeit einer temporalen Aussage)

Sei $\rho = (K, r)$ ein verteilter Ablauf eines Systems $\Sigma = (N, M_0)$ und seien β eine Belegung der Variablen von N und C ein Schnitt aus ρ.

(i) Wenn $\varphi \in \text{ZA}(N)$, dann gilt $\rho, C, \beta \models \varphi$ genau dann, wenn $r(C), \beta \models \varphi$.

(ii) Es gilt genau dann $\rho, C, \beta \models \Box\,\varphi$, wenn für jeden Schnitt $C' > C$ aus ρ gilt $r(C'), \beta \models \varphi$.

(iii) Es gilt genau dann $\rho, C, \beta \models (\varphi \wedge \varphi')$, wenn $\rho, C, \beta \models \varphi$ und $\rho, C, \beta \models \varphi'$.

(iv) Es gilt genau dann $\rho, C, \beta \models \neg\varphi$, wenn nicht $\rho, C, \beta \models \varphi$ gilt.

(v) Es gilt genau dann $\rho, C, \beta \models \exists x\varphi$, wenn es eine Belegung β' mit $\beta(y) = \beta'(y)$ für alle $y \neq x$, so daß $\rho, C, \beta \models \varphi$ gilt.

Eine temporale Aussage φ gilt im Ablauf ρ, wenn für jede Belegung β gilt: $\rho, {}^\circ\mathrm{K}, \beta \models \varphi$. Die Aussage gilt im System Σ (in Symbolen: $\Sigma \models \varphi$), wenn sie in jedem Ablauf gilt. $\star$

Die logischen Operatoren $\vee, \rightarrow, \leftrightarrow$ sind die üblichen Abkürzungen. Außerdem stehen

$$\begin{array}{rll} \forall x\varphi & \text{abkürzend für} & \neg\exists x\neg\varphi, \\ \Diamond\,\varphi & \text{abkürzend für} & \neg\,\Box\,\neg\varphi, \\ \boxtimes\,\varphi & \text{abkürzend für} & \Box(\varphi \rightarrow (\Box\,\varphi)), \\ \varphi \rhd \psi & \text{abkürzend für} & \Box(\varphi \rightarrow (\Diamond\,\psi)) \end{array}$$

Sei φ eine Zustandsaussage. Wenn $\Box\,\varphi$ in einem System gilt, dann nennen wir $\Box\,\varphi$ eine Invariante des Systems. In diesem Fall gilt φ in jeder erreichbaren Markierung des Systems. Besonders einfach zu handhabende Invarianten sind *Stelleninvarianten*. Eine Stelleninvariante ordnet jeder Markierung eines Systems einen Wert zu, der sich beim Schalten nicht verändert. In unserem Beispiel aus Abb. 4.1 gilt, daß jeder Agent immer entweder aktiv oder passiv ist. Wir beschreiben dies durch die Aussage

$$\forall x \in A \quad (\Box\,\mathsf{active}[x] + \mathsf{passive}[x] = 1).$$

Für eine Zustandsaussage φ nennen wir $\boxtimes\varphi$ eine stabile Aussage. Wenn $\boxtimes\varphi$ in Σ gilt, dann bedeutet das, wenn φ einmal gilt, dann gilt danach immer φ. Wenn in Σ_{vb} alle Agenten passiv sind und keine Nachrichten mehr unterwegs sind, wird der Zustand des Systems nicht mehr verändert. Wir drücken dies mit der folgenden Formel aus:

$$\boxtimes(\mathsf{passive} = A \wedge \mathsf{basic\ messages} = \emptyset).$$

Schließlich legen wir noch folgende Notation fest: Wenn p ein Stellenname ist, dann drücken wir mit $\mathsf{p}(a)$ aus, daß mindestens eine Marke a auf der Stelle liegt.

Wir haben in diesem Abschnitt temporale Aussagen über verteilten Abläufen eines Systems interpretiert. Ähnlich induktiv können wir eine temporale Aussage auch über den sequentiellen Abläufen eines Systems interpretieren (und mit $\Sigma \models_{\mathrm{seq}} \varphi$ bezeichnen). Wir definieren dazu zuerst die Gültigkeit einer Zustandsaussage in einer Markierung M und einer Belegung β, wie in Def. 4.12. Die Gültigkeit in einem sequentiellen Ablauf definieren wir dann wie in Def. 4.13, nur daß wir Markierungen statt Schnitte des Ablaufs verwenden.

Durch die Beziehung zwischen verteilten und sequentiellen Abläufen (Bem. 4.6) gilt in sicheren Systemen für jede temporale Aussage: Wenn $\Sigma \models_{\mathrm{seq}} \varphi$ gilt, dann gilt auch $\Sigma \models \varphi$. Die Umkehrung gilt nicht immer. Für viele temporale Aussagen gilt die Umkehrung aber, z.B. für Invarianten und stabile Aussagen. Wir wollen hier jedoch nicht genauer darauf eingehen.

5 Verifikation mit Verfeinerungen

5.1 Algebraische Transitionsverfeinerung

Wir definieren nun algebraische Transitionsverfeinerung analog zur simultanen Transitionsverfeinerung. Es ist nicht völlig kanonisch, was ein Ersetzungsnetz für eine algebraische Transition ist. Bevor wir formal ein algebraisches Ersetzungsnetz definieren, betrachten wir ein einfaches Beispiel.

Sei (A, N) ein Kommunikationsnetzwerk, das aus vier Agenten $A = \{a, b, c, d\}$ besteht, mit $N = \{(a, b), (b, a), (c, d), (d, c)\}$, d.h. a und b sind Nachbarn und c und d sind Nachbarn. In Abb. 5.1 ist ein Algorithmus für dieses Kommunikationsnetzwerk definiert.

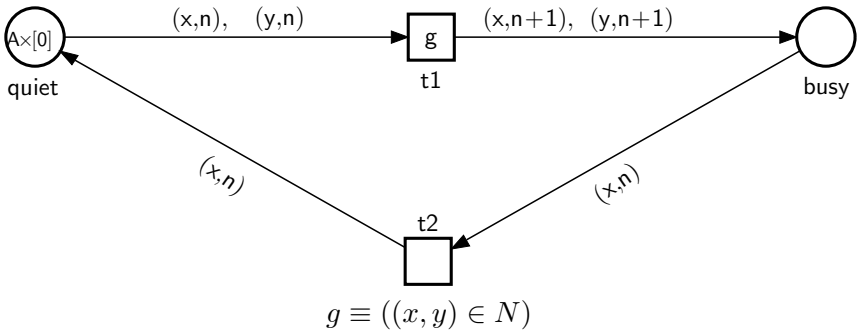


Abb. 5.1: Das System Σ

Zu Beginn haben alle Agenten die Rundennummer 0 und sind in Zustand quiet. Zwei Nachbarn mit der gleichen Rundennummer können gemeinsam vom Zustand quiet in die nächste Runde in den Zustand busy übergehen. Jeder Agent der im Zustand busy ist, kann seine Runde beenden, indem er in den Zustand quiet übergeht, ohne seine Rundennummer zu ändern.

Die Entfaltung der Transition $t1$ besteht aus einer Transition für jeden Modus, in dem die Aktivierungsbedingung von $t1$ zu true ausgewertet wird. Das sind vier Transitionen für jede Rundennummer k, nämlich für die Modi $[x = a, y = b, n = k]$, $[x = c, y = d, n = k]$, $[x = b, y = a, n = k]$ und $[x = d, y = a, n = k]$. Für alle anderen Belegungen der Variablen wird die Aktivierungsbedingung g zu false ausgewertet. In Abb. 5.2(links) haben wir für eine feste Rundennummer $n = k$ die Entfaltung der Transition $t1$ für zwei der vier Modi angegeben: für den Modus $[x = a, y = b, n = k]$ und $[x = c, y = d, n = k]$. Die Entfaltung für den Modus $[x = b, y = a, n = k]$ ist identisch mit der Entfaltung für $[x = a, y = b, n = k]$, nur die Transition ist mit $t1.ba$ statt mit $t1.ab$ beschriftet. Daneben haben wir für jedes der

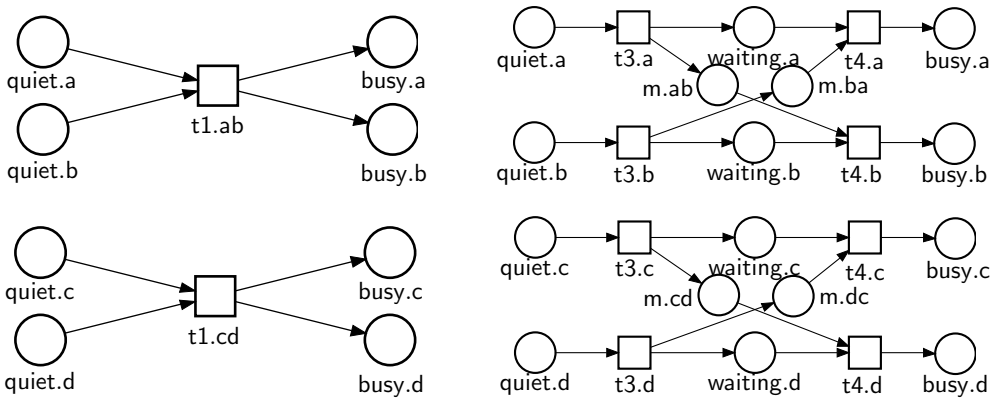
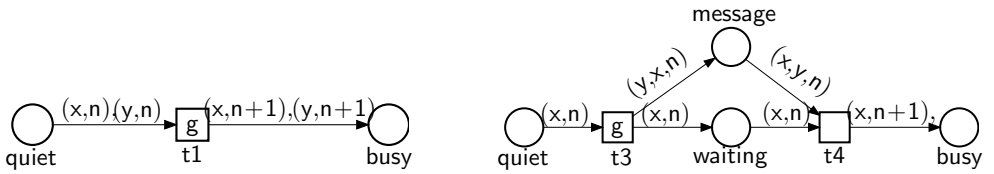


Abb. 5.2: Die Entfaltung von $t1$ und Ersetzungsnetze für den Modus $[n = k]$

beiden elementaren Petrinetze ein Ersetzungsnetz abgebildet. In jeder Stellen- und Transitionsbezeichnung dieser Abbildung kommt eigentlich die Rundennummer k vor, die wir aus Platzgründen weggelassen haben. Für verschiedene Rundennummern sind die Ersetzungsnetze disjunkt. Wir haben damit Ersetzungsnetze für alle Modi der Transition $t1$ angeben, so daß die Ersetzungsnetze für je zwei Modi disjunkt sind. Die beiden Petrinetze rechts sind auch Ersetzungsnetze für die Entfaltungen der nicht abgebildeten Modi$[x = b, y = a, n = k]$, $[x = d, y = a, n = k]$.

Ein algebraisches Ersetzungsnetz für die Transition $t1$ ist ein algebraisches Petrinetz, dessen Entfaltung gerade die Vereinigung einer Menge von Ersetzungsnetzen für die Modi der entfalteten Transition ist. Die Vereinigung zweier Netze (P_1, T_1, F_1) und (P_2, T_2, F_2) definieren wir als $N_1 \cup N_2 = (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2)$. Außerdem sollen die Algebra und die Sorten der Stellen des Vor- und Nachbereichs von $t1$ im algebraischen Ersetzungsnetz identisch mit denen von Σ sein.



(a) Transition $t1$

(b) Ersetzungsnetz N_E

Abb. 5.3: Aktivierungsbedingung $g \equiv ((x, y) \in N)$

In Abb. 5.3 sind noch einmal die Transition $t1$ und daneben ein algebraisches Ersetzungsnetz N_E für $t1$ dargestellt. Die Entfaltung von N_E ist genau die Vereinigung der beiden Ersetzungsnetze für jeden Modus $[n = k]$ aus Abb. 5.2.

Genau wie ein algebraisches Petrinetz, hat auch jede Transition eines algebraischen Petrinetzes eine Entfaltung. Diese Entfaltung besteht aus vielen elementaren Transitionen; eine Transition für jeden Modus, in dem die Aktivierungsbedingung der Transition zu true ausgewertet wird (vgl. Def. 4.7). Auch jede Aktion (t, β), deren

Aktivierungsbedingung zu true ausgewertet wird, hat eine Entfaltung. Diese besteht nur aus einer einzigen Transition. Die Entfaltung der Transition t ist genau die Vereinigung der Entfaltungen aller Aktionen (t, β), für die die Aktivierungsbedingung von t zu true ausgewertet wird.

Wir betrachten nun eine Transition t eines beliebigen Systems Σ. Wenn wir die algebraische Transition t in Σ ersetzen, ist dies gleichbedeutend damit, daß wir in der Entfaltung Σ^e von Σ simultan für alle Belegungen β die Transitionen (t, β) ersetzen. Sei nun $\mathcal{N}_E = \{N_{(t,\beta)} \mid (t, \beta) \in T^e\}$ eine Menge von Ersetzungsnetzen für alle Transitionen (t, β) in der Entfaltung Σ^e. Das algebraische Petrinetz N_E ist ein *algebraisches Ersetzungsnetz* für t in Σ, wenn die Entfaltung von N_E gerade die Vereinigung $\bigcup_{(t,\beta)\in T^e} N_{(t,\beta)}$ der Ersetzungsnetze der Aktionen von t in der Entfaltung ist. Damit N_E in das System Σ eingesetzt werden kann, geben wir noch einige syntaktische Restriktionen an.

Definition 5.1 (Algebraisches Ersetzungsnetz)

Seien $N = ((P, T, F), \mathcal{A}, \mathcal{X}, d, w, g)$ ein algebraisches Petrinetz und t eine Transition aus N. Sei außerdem N^e die Entfaltung von N. Das algebraische Petrinetz $N_E = ((P_E, T_E, F_E), \mathcal{A}_E, \mathcal{X}_E, d_E, w_E, g_E)$ ist ein *algebraisches Ersetzungsnetz* für t in N, wenn es eine Menge $\mathcal{N}_E = \{N_{(t,\beta)} \mid (t, \beta) \in T^e\}$ von Ersetzungsnetzen für alle Transitionen (t, β) in der Entfaltung N^e gibt, so daß die Entfaltung von N_E gerade die Vereinigung $\bigcup_{(t,\beta)\in T^e} N_{(t,\beta)}$ der Ersetzungsnetze der Aktionen von t in der Entfaltung ist und außerdem gilt:

(i) $P \cap P_E = {}^\bullet t \cup t^\bullet$ und $T \cap T_E = \emptyset$,

(ii) $\mathcal{A}_E = \mathcal{A}$ und $\mathcal{X}_E = \mathcal{X}$,

(iii) für alle $p \in {}^\bullet t \cup t^\bullet$ gilt $d(p) = d_E(p)$, ⋆

In algebraischen Petrinetzen ist es schwieriger als in elementaren Petrinetzen festzustellen, ob ein Petrinetz ein Ersetzungsnetz für eine Transition ist. Das folgende Lemma gibt ein hinreichendes Kriterium dafür an, wann ein algebraisches Netz ein algebraisches Ersetzungsnetz für eine Transition ist.

Lemma 5.2 Seien $N = ((P, T, F), \mathcal{A}, \mathcal{X}, d, w, g)$ ein algebraisches Petrinetz und t eine Transition aus N. Das algebraische Petrinetz
$N_E = ((P_E, T_E, F_E), \mathcal{A}_E, \mathcal{X}_E, d_E, w_E, g_E)$ ist ein algebraisches Ersetzungsnetz für t in N, wenn

(i) $P \cap P_E = {}^\bullet t \cup t^\bullet$ und $T \cap T_E = \emptyset$,

(ii) $\mathcal{A}_E = \mathcal{A}$ und $\mathcal{X}_E = \mathcal{X}$,

(iii) für alle $p \in {}^\bullet t \cup t^\bullet$ gilt $d(p) = d_E(p)$ und

(iv) für jeden Modus β ist jeder Ablauf (K, r) von $(N_E, (t, \beta)^-)$ endlich und $r(\mathrm{K}^\circ) = (t, \beta)^+$. ⋆

Beweis: Wir müssen zeigen, daß die Entfaltung von N_E die Vereinigung einer Familie von Ersetzungsnetzen der entfalteten Modi der Transition ist. Nach Satz 4.8

ist die Entfaltung N_E^e von N_E ein Ersetzungsnetz für jede Transition (t, β) der Entfaltung Σ^e von Σ. Wir wählen also zu jedem Modus β das Netz $N_{(t,\beta)} = N_E^e$ als Ersetzungsnetz. Die Vereinigung dieser Ersetzungsnetze ist wieder N_E^e. q.e.d.

Wir markieren N_E nun mit dem Vorbereich von $t1$ für die Belegung $[x = a, y = b, n = 0]$, also mit den Marken $(a, 0), (b, 0)$ auf der Stelle quiet. Mit dieser Anfangsmarkierung hat N_E nur einen Ablauf, der mit den Marken $(a, 1), (b, 1)$ auf der Stelle busy endet. Dies ist gerade der Nachbereich der Transition $t1$ im Modus $[x = a, y = b, n = 0]$. Um zu beweisen, daß N_E ein algebraisches Ersetzungsnetz von $t1$ ist, müssen wir diese Überlegung für jeden Modus anstellen. Das ist meist viel einfacher, als anhand der Definition zu zeigen, daß N_E ein Ersetzungsnetz ist.

Das erweiterte System, die Expansion eines Ablaufs und algebraische Transitionsverfeinerung werden nun kanonisch definiert.

Definition 5.3 (Erweitertes System)

Sei $N = ((P, T, F), \mathcal{A}, \mathcal{X}, d, w, g)$ ein algebraisches Petrinetz, t eine Transition aus N und $N_E = ((P_E, T_E, F_E), \mathcal{A}, \mathcal{X}, d_E, w_E, g_E)$ ein algebraisches Ersetzungsnetz für t. Dann ist das *erweiterte* Petrinetz $N\big[t \to N_E\big]$ definiert durch

(i) das Netz (P', T', F') mit $P' = P \cup P_E$, $T' = (T \setminus \{t\}) \cup T_E$ und $F' = (F \cup F_E) \cap (P' \times T' \cup T' \times P')$.

(ii) die Algebra $\mathcal{A}$,

(iii) die sortierte Variablenmenge $\mathcal{X}$,

(iv) die Sortenfunktion $d' = d \cup d_E$

(v) die Gewichtungsfunktion $w' = w|_{F \cap F'} \cup w_E$

(vi) die Aktivierungsfunktion $g' = g|_{T \setminus \{t\}} \cup g_E$.

Für ein System Σ bezeichnen wir mit $\Sigma\big[t \to N_E\big]$ das *erweiterte System* $(N\big[t \to N_E\big], M_0')$, wobei $M_0'(p) = M_0(p)$ für $p \in P$ und $M_0'(p) = [\,]$ für alle anderen p. $\star$

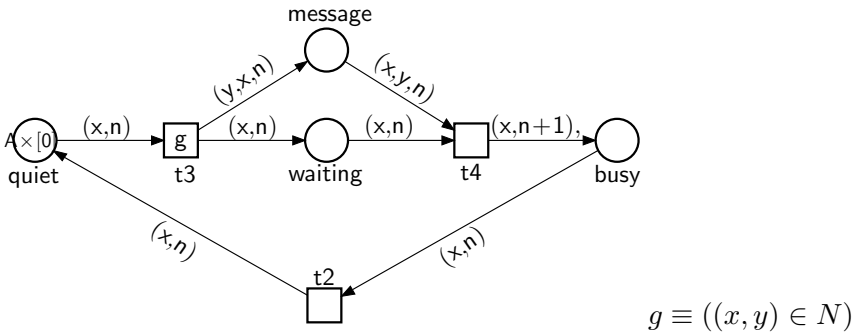


Abb. 5.4: Das System $\Sigma\big[t1 \to N_E\big]$

In Abb. 5.4 haben wir das erweiterte System für das obige Beispiel dargestellt.

Definition 5.4 (Expansion eines Ablaufs)

Wir betrachten ein System $\Sigma = (N, M_0)$ mit $N = ((P, T, F), \mathcal{A}, \mathcal{X}, d, w, g)$, eine Transition t aus N und ein Ersetzungsnetz $N_E = ((P_E, T_E, F_E), \mathcal{A}, \mathcal{X}, d_E, w_E, g_E)$ für t. Sei $\rho = ((B, E, \lessdot), r)$ ein Ablauf von Σ. Wir definieren

(i) $E(t) = \{a \in E \mid$ es gibt einen Modus β mit $r(a) = (t, \beta)\}$ die Menge der Ereignisse aus ρ, die mit dem Schalten der Transition t assoziiert werden,

(ii) zu jedem $a \in E(t)$ einen Ablauf (K_a, r_a) von $(N_E, r(a)^-)$ mit ${}^\circ\mathrm{K}_a = {}^\bullet a$ und $\mathrm{K}_a^\circ = a^\bullet$ und für alle $b \in {}^\bullet a \cup a^\bullet$ gilt $r(b) = r_a(b)$, so daß die Mengen $E_a \cup B_a \setminus ({}^\circ\mathrm{K}_a \cup \mathrm{K}^\circ{}_a)$ paarweise disjunkt für verschiedene a und außerdem paarweise disjunkt zu $E \cup B$ sind

(iii) das Kausalnetz $(B', E', \lessdot')$ als

$$\begin{aligned} B' &= B \cup \bigcup\{B_a \mid a \in E(t)\}, \\ E' &= (E \setminus E(t)) \cup \bigcup\{E_a \mid a \in E(t)\}, \\ \lessdot' &= \big(\lessdot \setminus (E(t) \times B \cup B \times E(t))\big) \cup \bigcup\{\lessdot_a \mid a \in E(t)\}, \end{aligned}$$

(iv) die $\Sigma\big[t \to N_E\big]$-Beschriftung r' definiert durch

$$r'(x) = \begin{cases} r(x) & \text{für } x \in B \cup E \setminus E(t), \\ r_a(x) & \text{für } x \in B_a \cup E_a. \end{cases}$$

Dann nennen wir $\rho' = ((B', E', \lessdot'), r')$ eine $\big[t \to N_E\big]$-*Expansion* von ρ. ⋆

Definition 5.5 (Algebraische Transitionsverfeinerung)

Seien Σ ein System und t eine Transition aus Σ. N_E ist eine *algebraische Transitionsverfeinerung* der Transition t im System Σ, wenn N_E ein algebraisches Ersetzungsnetz ist und die Menge der verteilten Abläufe von $\Sigma\big[t \to N_E\big]$ gleich der Menge der $\big[t \to N_E\big]$-Expansionen der verteilten Abläufe von Σ ist. ⋆

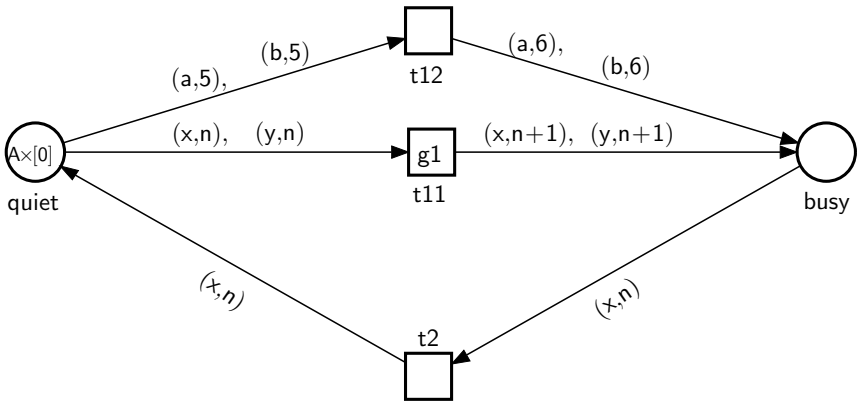
Wenn aus dem Kontext hervorgeht, worum es sich handelt, werden wir das Adjektiv *algebraische* weglassen.

Auch algebraische Transitionsverfeinerung kann man kanonisch, wie im elementaren Fall, auf die simultane Verfeinerung mehrerer algebraischer Transitionen erweitern. Zwischen algebraischer und simultaner algebraischer Transitionsverfeinerung unterscheiden wir konzeptionell nicht, denn beide sind äquivalent zu einer simultanen elementaren Transitionsverfeinerung.

Beweiskriterien für algebraische Transitionsverfeinerung

Durch unsere Definition eines algebraischen Ersetzungsnetzes, können wir die Beweiskriterien für simultane Transitionsverfeinerung auf algebraische Transitionsverfeinerung übertragen. Wir betrachten dazu noch einmal das Beispiel aus dem letzten Abschnitt. Wir wollen nun zeigen, daß N_E eine Transitionsverfeinerung von $t1$ in Σ

ist. Um Satz 3.7 anzuwenden, müssen wir die Modi von $t1$ so in Äquivalenzklassen aufteilen können, daß die Entfaltung des Ersetzungsnetzes in disjunkte Netze zerfällt, für jede Äquivalenzklasse ein Netz. Für unser Beispiel ist das einfach. Die Transition $t1$ hat für jede Rundennummer n vier Modi, in denen sie schalten kann, von denen je zwei das gleiche Ersetzungsnetz aus Abb. 5.2 haben. Diese beiden Modi bilden eine Äquivalenzklasse. Die anderen beiden Modi, mit dem anderen Ersetzungsnetz bilden die zweite Äquivalenzklasse. Wir haben also für jede Rundennummer zwei Äquivalenzklassen.



$$g1 \equiv ((x, y) \in N \wedge \{x, y\} \neq \{a, b\} \wedge n \neq 5)$$

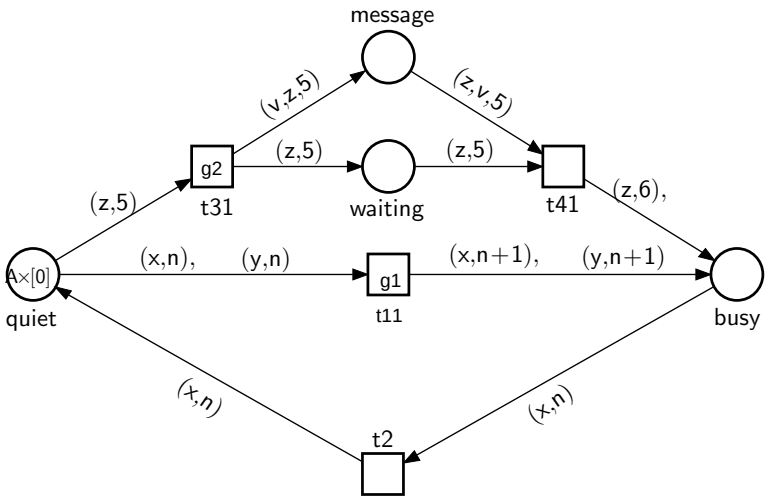
Abb. 5.5: Das System Σ'

Wir können nun eine beliebige Äquivalenzklasse aus der Transition $t1$ herauslösen. In Abb. 5.5 haben wir dies exemplarisch für die Klasse aus den beiden Modi $[x = a, y = b, n = 5]$ und $[x = b, y = a, n = 5]$ getan. Die Transition $t12$ simuliert hier die Transition $t1$ im Modus $[x = a, y = b, n = 5]$ und $[x = b, y = a, n = 5]$, die Transition $t11$ simuliert die Transition $t1$ in allen anderen Modi. Die Systeme Σ und Σ' haben die gleichen verteilten Abläufe, bis auf die Beschriftung der Transitionen.

Wir ersetzen nun nur die Transition $t12$, wie in Abb. 5.6 angegeben. Wenn wir dann für beliebige Modi zeigen können, daß diese Ersetzung eine Transitionsverfeinerung in Σ' ist, können wir mit Satz 3.7 folgern, daß das algebraische Ersetzungsnetz N_E eine algebraische Transitionsverfeinerung von $t1$ in Σ ist.

Dieses Verfahren ist umständlich, aber nicht kompliziert. Den Vorteil, den wir davon haben ist der folgende: Wir müssen jetzt nicht mehr für alle Modi simultan beweisen, daß das Ersetzungsnetz eine Transitionsverfeinerung, sondern immer nur für eine Klasse von Modi. Das ist im allgemeinen einfacher.

In unserem Beispiel ist es leicht zu zeigen, daß die Transitionen $t31$ und $t41$ in jedem Ablauf von $\Sigma'[t12 \rightarrow N'_E]$ genau einmal im Modus $[z = a]$ und einmal im Modus $[z = b]$ auftreten und diese Ereignisse zusammen einen Ablauf des Ersetzungsnetzes formen. Wir können davon ausgehen, daß das System bis zur Runde 4 genau wie das System Σ arbeitet und Eigenschaften von Σ im Beweis verwenden.



$$g1 \equiv ((x,y) \in N \wedge \{x,y\} \neq \{a,b\} \wedge n \neq 5)$$
$$g2 \equiv (z \in \{a,b\} \wedge (z,v) \in N)$$

Abb. 5.6: Das System $\Sigma'\big[t12 \rightarrow N'_E\big]$

5.2 Bewahrung von Eigenschaften bei Transitionsverfeinerung

In diesem Abschnitt diskutieren wir, welche Eigenschaften bei einer Transitionsverfeinerung erhalten bleiben. In [AS85] zeigen Alpern und Schneider, daß man jedes System durch Sicherheits- und Lebendigkeitseigenschaften spezifizieren kann. Eine Sicherheitseigenschaft beschreibt intuitiv, daß nie "etwas Schlechtes" passiert. Eine Lebendigkeitseigenschaft beschreibt, daß irgendwann "etwas Gutes" passiert. Wenn φ eine Zustandsaussage ist, dann ist die Invariante $\Box\,\varphi$ eine typische Sicherheitseigenschaft und $\Diamond\,\varphi$ ist eine typische Lebendigkeitseigenschaft.

Wir zeigen, daß Lebendigkeitseigenschaften der Form $\Diamond\,\varphi$ bei einer Transitionsverfeinerung erhalten bleiben und Sicherheitseigenschaften der Form $\Box\,\varphi$ unter gewissen Einschränkungen erhalten bleiben.

Eine Zustandsaussage gilt in einem System genau dann, wenn sie in der Anfangsmarkierung gilt. Da ein System und das verfeinerte System dieselbe Anfangsmarkierung haben, stellen wir zuerst folgendes fest:

Bemerkung 5.6 Seien Σ ein System, t eine Transition aus Σ und N_E eine Transitionsverfeinerung für t in Σ. Eine Zustandsaussage über Σ gilt in Σ genau dann, wenn sie in $\Sigma\big[t \rightarrow N_E\big]$ gilt. $\star$

Satz 5.7 Seien Σ ein System, t eine Transition aus Σ und N_E eine Transitionsverfeinerung für t in Σ. Sei φ eine Zustandsaussage über Σ.

(i) Wenn $\Sigma \models \Diamond\,\varphi$, dann gilt auch $\Sigma\big[t \rightarrow N_E\big] \models \Diamond\,\varphi$.

(ii) Wenn $\Sigma \models \Box\,\Diamond\,\varphi$, dann gilt auch $\Sigma\big[t \rightarrow N_E\big] \models \Box\,\Diamond\,\varphi$. $\star$

Bevor wir den Satz beweisen, stellen wir noch eine Beziehung zwischen den Schnitten in den Abläufen beider Systeme her, die wir zum Beweis benötigen. Die folgende Bemerkung folgt direkt aus der Definition 5.4.

Bemerkung 5.8 Seien Σ ein System, t eine Transition aus Σ und N_E eine Transitionsverfeinerung für t in Σ. Wenn $\rho = (K, r)$ ein Ablauf von Σ ist und $\rho' = (K', r')$ eine $\big[t \to N_E\big]$-Expansion von ρ ist, dann ist jeder Schnitt C von K auch ein Schnitt in K' und es gilt $r(C) = r'(C)$. ⋆

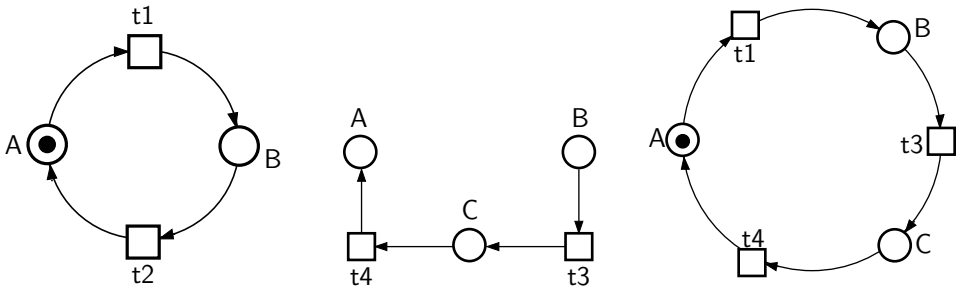
Beweis: (von Satz 5.7) Sei $\rho' = (K', r')$ ein beliebiger Ablauf von $\Sigma\big[t \to N_E\big]$. Da N_E eine Transitionsverfeinerung von t in Σ ist, gibt es einen Ablauf $\rho = (K, r)$ von Σ, so daß ρ' eine $\big[t \to N_E\big]$-Expansion von ρ ist.

Zu (i): Wir müssen zeigen, daß es in ρ' einen Schnitt C gibt, so daß $r'(C) \models \varphi$. Nach Voraussetzung gibt es in ρ einen Schnitt C, so daß $r(C) \models \varphi$. Wegen Bemerkung 5.8 ist C auch ein Schnitt in ρ' und es gilt $r'(C) \models \varphi$.

Zu (ii): Wir müssen zeigen, daß es in ρ' zu jedem Schnitt C einen Schnitt C' gibt, so daß $r'(C') \models \varphi$. Der Beweis ist analog zum Beweis von (i). q.e.d.

Satz 5.9 Seien Σ ein System, t eine Transition aus Σ und N_E eine Transitionsverfeinerung für t in Σ. Sei Φ eine temporale Aussage aus Termen über Σ, in der nur die Operatoren $\wedge$ und $\Diamond$ vorkommen. Wenn $\Sigma \models \Phi$, dann gilt auch $\Sigma\big[t \to N_E\big] \models \Phi$. ⋆

Beweis: Der Beweis ist ähnlich wie der Beweis von Satz 5.7, nur mit einer Induktion über den Aufbau von Formeln aus $\wedge$ und $\Diamond$. q.e.d.



(a) Σ (b) N_E (c) $\Sigma\big[t2 \to N_E\big]$

Abb. 5.7: Beispiel für Zerstörung von Invarianten

Invarianten bleiben bei einer Transitionsverfeinerung meist nicht erhalten. Wir betrachten dazu das Beispiel in Abb. 5.7(a). In Σ gilt die Invariante $\Box\,\mathsf{A} + \mathsf{B} = 1$. Wenn wir die Transition $t2$ durch die Hintereinanderausführung von $t3$ und $t4$ ersetzen (Abb. 5.7(b)), dann ist diese Ersetzung eine Verfeinerung, aber die Invariante gilt nicht mehr. Die Formel $\mathsf{A} + \mathsf{B} = 1$ gilt immer dann nicht, wenn $t3$, aber noch nicht $t4$ geschaltet hat. Dann hat das Ersetzungsnetz einen Ablauf begonnen, aber noch nicht beendet. Wenn in einem Schnitt C in einem Ablauf von $\Sigma\big[t2 \to N_E\big]$

mindestens eine Aktion des Ersetzungsnetzes vor C liegt und mindestens eine Aktion aus dem gleichen Ablauf des Ersetzungsnetzes nach C, dann nennen wir das Ersetzungsnetz *aktiv* im Schnitt C.

Oft ist es einfach, eine Zustandsaussage Θ zu finden, die genau dann gilt, wenn das Ersetzungsnetz nicht aktiv ist. Wenn Θ gilt, ist man also in einem Zustand, der auch im Ausgangsnetz seine Entsprechung hat. Wir nennen dies einen *zulässigen Zustand* des verfeinerten Systems. Etwas formaler: Seien $\rho = (K, r)$ ein Ablauf von Σ und $\rho' = (K', r')$ eine $\big[t \to N_E\big]$-Expansion von ρ. Für einen Schnitt C aus K' gilt $r'(C) \models \Theta$ genau dann, wenn C auch ein Schnitt aus K ist. Nach dieser Definition und mit Bemerkung 5.8 gilt $\Sigma \models \Box\,\Theta$. Wenn $\Box\,\varphi$ mit $\varphi \in \mathrm{ZA}(\Sigma)$ eine Invariante von Σ ist, dann können wir zeigen, daß in $\Sigma\big[t \to N_E\big]$ zumindest immer dann φ gilt, wenn Θ gilt. In unserem Beispiel ist das Ersetzungsnetz nicht aktiv, wenn C nicht markiert ist. Wir können daher schließen: $\Box(\mathsf{C} = 0 \to \mathsf{A} + \mathsf{B} = 1)$.

Im Beispiel von Seite 74 ist das Ersetzungsnetz genau dann nicht aktiv, wenn keine Nachricht auf der Stelle messages liegt. Im Ausgangssystem Σ gilt, daß benachbarte Agenten immer die gleiche Rundennummer haben. Im verfeinerten System gilt also: Immer wenn keine Nachricht auf der Stelle messages liegt, dann haben benachbarte Agenten die gleiche Rundennummer.

Wenn wir eine solche Aussage Θ gefunden haben, können wir sogar viel mehr über das verfeinerte System sagen:

Satz 5.10 Seien Σ ein System, t eine Transition aus Σ und N_E eine Transitionsverfeinerung für t in Σ. Sei Θ eine Zustandsaussage über $\Sigma\big[t \to N_E\big]$, die genau dann gilt, wenn das Ersetzungsnetz nicht aktiv ist. Sei φ eine Zustandsaussage über Σ. Dann gilt

$$\Sigma \models \Box\,\varphi \qquad \text{genau dann, wenn} \qquad \Sigma\big[t \to N_E\big] \models \Box(\Theta \to \varphi)$$
$$\Sigma \models \Diamond\,\varphi \qquad \text{genau dann, wenn} \qquad \Sigma\big[t \to N_E\big] \models \Diamond(\varphi \wedge \Theta)$$

$\star$

Beweis: Wir beweisen nur die erste Behauptung des Satzes. Der Beweis der zweiten Behauptung ist analog. Sei $\rho' = (K', r')$ ein beliebiger Ablauf von $\Sigma\big[t \to N_E\big]$. Da N_E eine Transitionsverfeinerung von t in Σ ist, gibt es einen Ablauf $\rho = (K, r)$ von Σ, so daß ρ' eine $\big[t \to N_E\big]$-Expansion von ρ ist.

Wir müssen zeigen, daß φ genau dann in jedem Schnitt von ρ gilt, wenn $\Theta \to \varphi$ in jedem Schnitt von ρ' gilt. Dies ist der Fall, wegen Bemerkung 5.8 und da jeder Schnitt C von ρ' genau dann ein Schnitt von ρ ist, wenn $r'(C) \models \Theta$.

Da wir jeden Ablauf von Σ als Umkehrung einer $\big[t \to N_E\big]$-Expansion eines Ablaufs von $\Sigma\big[t \to N_E\big]$ erhalten können, ist die erste Behauptung damit bewiesen. q.e.d.

Zusammen mit Bemerkung 5.6 können wir damit induktiv viele weitere Eigenschaften von $\Sigma\big[t \to N_E\big]$ nach Bedarf ableiten.

Manchmal ist es schwer eine Formel Θ zu finden, die genau die zulässigen Zustände beschreibt. Dann kann man sich auch mit einer restriktiveren Formel $\bar{\Theta}$ begnügen, so daß immer wenn $\bar{\Theta}$ gilt, das Ersetzungsnetz nicht aktiv ist (aber nicht die Umkehrung gilt). Auch dann bleiben Eigenschaften wie oben erhalten, aber die Umkehrung gilt nicht.

Satz 5.11 Seien Σ ein System, t eine Transition aus Σ und N_E eine Transitionsverfeinerung für t in Σ. Sei $\bar{\Theta}$ eine Zustandsaussage über $\Sigma\big[t \to N_E\big]$ aus der folgt, daß das Ersetzungsnetz nicht aktiv ist. Sei φ eine Zustandsaussage über Σ. Dann gilt

$$\begin{array}{llll} \text{wenn} & \Sigma \models \Box\,\varphi, & \text{dann} & \Sigma\big[t \to N_E\big] \models \Box(\bar{\Theta} \to \varphi) \\ \text{wenn} & \Sigma \models \Diamond\,\varphi, & \text{dann} & \Sigma\big[t \to N_E\big] \models \Diamond(\varphi \wedge \bar{\Theta}) \end{array}$$

$\star$

Beweis: Analog zum Beweis von Satz 5.10. q.e.d.

Wir nennen M eine Endmarkierung des Systems, wenn es einen endlichen Ablauf (K, r) gibt, so daß $M = r(\mathrm{K}^\circ)$.

Folgerung 5.12 Wenn sowohl das Ausgangssystem, als auch das verfeinerte System nur endliche Abläufe haben, dann haben beide Systeme die gleichen Endmarkierungen. $\star$

5.3 Datenerweiterung

Um einen korrekten abstrakten Algorithmus zu einem implementierbaren Algorithmus zu verfeinern, werden oft neue Variablen eingeführt, die den Agenten lokal zugeordnet werden. Wir nennen dies eine *Datenerweiterung*. Eine Datenerweiterung ist zunächst eine rein syntaktische Einführung neuer Variablen. Die neuen Variablen können zusätzlich synchronisieren, so daß das erweiterte System zu früh blockiert. Bei einer Datenerweiterung bleiben deshalb Lebendigkeitseigenschaften nicht notwendigerweise erhalten.

Wir nennen eine Datenerweiterung *korrekt*, wenn alle temporalen Eigenschaften des Algorithmus erhalten bleiben. Bei einer korrekten Datenerweiterung darf durch die neuen Variablen zusätzliche Synchronisation eingeführt werden, jedoch nur so, daß die Lebendigkeitseigenschaft des Systems nicht verletzt wird. Technisch gesehen, werden durch eine korrekte Datenerweiterung verteilte Abläufe ausgeschlossen; das erweiterte System hat weniger Abläufe als das Ausgangssystem. Ein Ablauf des erweiterten Systems bricht aber nicht früher ab, als ein Ablauf des Ausgangssystems.

Es ist bei einer korrekten Datenerweiterung nicht möglich, daß ein System durch die neuen Variablen so stark synchronisiert wird, daß überhaupt keine Aktion mehr stattfinden kann. Dann hat das System immer noch einen Ablauf, nämlich den, der nur aus dem Anfangszustand besteht. Dieser Ablauf erfüllt nicht die Lebendigkeitseigenschaft des Systems (es sei denn, auch im Ausgangssystem passiert nie etwas).

Korrekte Datenerweiterung ist vergleichbar mit dem Begriff *Datenverfeinerung*. Datenverfeinerung wird meist für sequentielle Abläufe von Systemen definiert. Einen Überblick über Literatur zur Datenverfeinerung (data refinement) findet man in [dRE98]. Wir benutzen einen anderen Begriff, da wir verteilte Abläufe zugrunde legen. Zwei Systeme, die dieselben sequentiellen Abläufe haben, haben nicht notwendigerweise dieselben verteilten Abläufe (siehe Beispiel auf Seite 63). Deshalb ist korrekte Datenerweiterung etwas restriktiver als Datenverfeinerung. Trotzdem ist

korrekte Datenerweiterung nicht wirklich neu oder interessant. Wir führen den Begriff nur ein, um besonders einfache Verfeinerungsschritte, in denen verteilte Abläufe erhalten bleiben, effizient behandeln zu können.

Da wir Datenerweiterung streng syntaktisch definieren, können wir ein einfaches Kriterium dafür ableiten, wann eine Datenerweiterung korrekt ist.

Ein Spezialfall der korrekten Datenerweiterung ist die *konservative Datenerweiterung* bei der die neue Variable das Verhalten des Algorithmus überhaupt nicht beeinflußt. Durch ein hinreichendes syntaktisches Kriterium kann man direkt am Petrinetzmodell ablesen, ob die neue Variable diese Eigenschaft hat. Für sequentielle Abläufe wurde dieser Fall z.B. in [Gri77] als *auxiliary variable axiom* beschrieben. Damit wird die Einführung von Hilfsvariablen, wie z.B. *history* und *prophecy* Variablen [AL91] als Hilfsmittel zur Verifikation gerechtfertigt.

Bevor wir Datenerweiterung definieren, werden wir zur Motivation zwei einfache Beispiele diskutieren.

Beispiel: Zwei Agenten

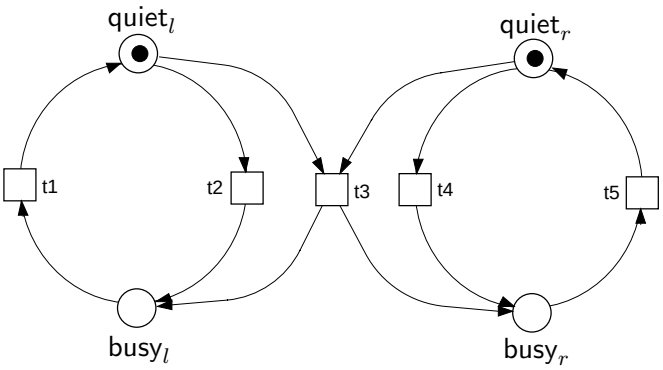


Abb. 5.8: Zwei Agenten

Wir betrachten ein System mit zwei Agenten, die lokal in Runden arbeiten (siehe Abb. 5.8). Jeder Agent durchläuft in jeder Runde die Zustände quiet und busy. Ein ruhiger Agent kann entweder eine lokale Aktion ausführen (Transitionen t2 oder t4) oder eine Aktion gemeinsam mit dem anderen Agenten ausführen, um in den Zustand busy überzugehen (Transition t3). Die Runden beider Agenten sind nicht synchronisiert, d.h. es kann passieren, daß beide Agenten ihre erste gemeinsame Aktion schalten, nachdem der linke Agent bereits fünfmal, der rechte dagegen erst dreimal busy war.

Temporale Eigenschaften dieses Systems sind zum Beispiel:

$$\Box \mathsf{quiet}_l + \mathsf{busy}_l = 1$$
$$\Box \mathsf{quiet}_r + \mathsf{busy}_r = 1$$

(Jeder Agent ist immer entweder quiet oder busy.)

und $\mathsf{busy}_l \rhd \mathsf{quiet}_l$ $\mathsf{busy}_r \rhd \mathsf{quiet}_r$

$\mathsf{quiet}_l \rhd \mathsf{busy}_l$ $\mathsf{quiet}_r \rhd \mathsf{busy}_r$

(Jeder Agent der busy ist, wird wieder quiet, und jeder Agent der quiet ist, wird wieder busy.)

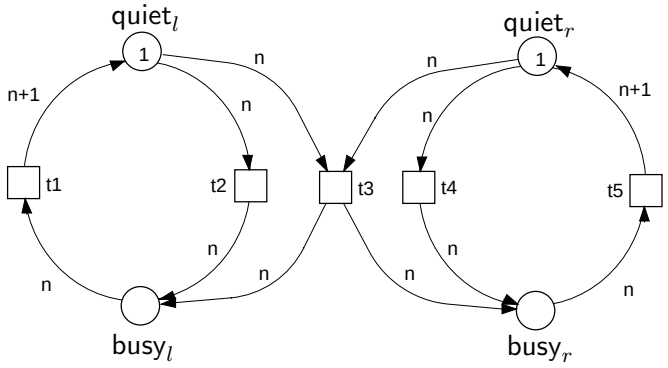


Abb. 5.9: Zwei Agenten

Eine Synchronisation der beiden Agenten kann durch eine zusätzliche Variable für jeden Agenten erreicht werden, die angibt, in welcher Runde sich der Agent gerade befindet. Dies wird im Modell in Abb. 5.9 durch die Variable n erreicht, die immer den Wert einer natürlichen Zahl hat und zu Beginn für beide Agenten gleich 1 ist. Durch die Inschriften an den eingehenden Kanten der Transition t3 ist gesichert, daß die Agenten nur noch dann gemeinsam schalten können, wenn sie in der gleichen Runde sind. Durch die Synchronisation mit der neuen Variablen werden Abläufe ausgeschlossen. Jeder Ablauf dieses neuen Systems ist aber ein Ablauf des alten Systems, wenn man die Rundennummern ausblendet. Deshalb bleibt die Gültigkeit aller temporalen Aussagen des alten Systems erhalten. Insbesondere kann auch dieses System nicht verklemmen, da die Lebendigkeitseigenschaft nicht verletzt wird.

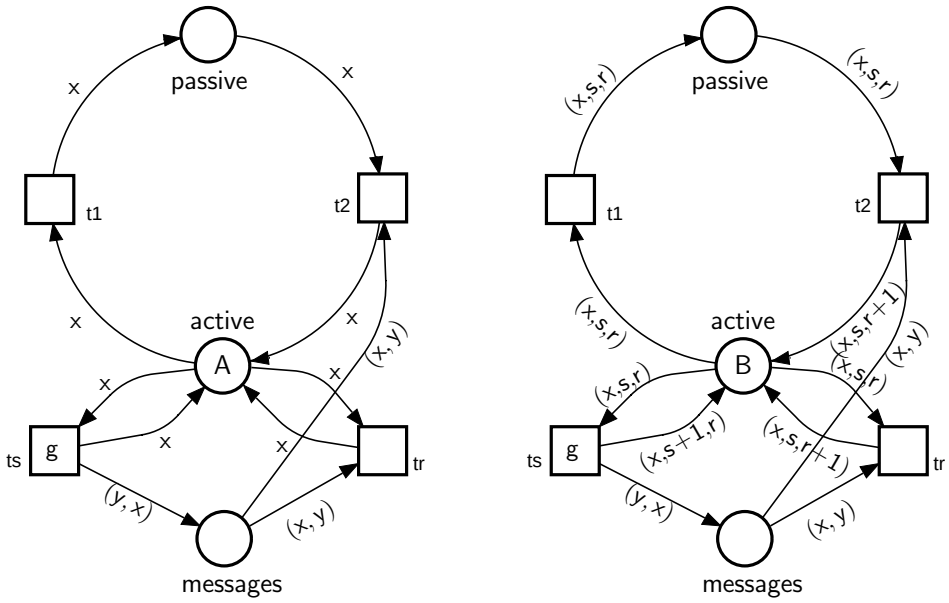
Beispiel: Verteilte Basisberechnung

Wir betrachten noch einmal das Beispiel der verteilten Berechnung (Seite 57).

In den meisten Algorithmen zur verteilten Feststellung der Terminierung des Systems muß jeder Agent zählen, wieviele Nachrichten er verschickt und empfangen hat. Er benutzt zwei neue Variablen: s, für die Anzahl der versendeten Nachrichten, r für die Anzahl der empfangenen Nachrichten. Das Zählen kann auf verschiedene Arten modelliert werden. Eine Möglichkeit ist in Abb. 5.10(b) dargestellt.

Dabei werden die neuen Variablen als Daten an die Marke angehängt, die den Zustand eines Agenten beschreibt. Anfangs setzen wir alle Zähler auf null, d.h. die Anfangsmarkierung der Stelle active ist $B = A \times \{(0,0)\}$.

Intuitiv wird man sagen, daß sich das Verhalten durch die neuen Variablen nicht verändert. Die Zähler synchronisieren den Algorithmus nicht zusätzlich und verhindern nie das Schalten einer Transition.



(a) ohne Zähler

(b) mit Zähler

Abb. 5.10: Verteilte Berechnung

Formal gesehen, liegen den Abläufen der beiden Systeme die gleichen Kausalnetze zugrunde, nur werden bei der Beschriftung der Abläufe des ersten Systems die Variablen s und r ausgeblendet. Insbesondere ist es unerheblich, mit welchen Werten die neuen Variablen anfangs belegt sind. Wir können den Zählern initial auch andere Werte als null geben. Diese Erweiterung ist eine konservative Datenerweiterung.

Datenerweiterung

Nun beschreiben wir formal, wie neue Variablen in ein bestehendes System eingebunden werden können. Dabei achten wir noch nicht darauf, ob sich das System mit diesen neuen Variablen immer noch korrekt verhält. Dieses rein syntaktische Hinzufügen von Variablen nennen wir *Datenerweiterung*.

Sei $\mathcal{A} = (U, Op)$ eine Algebra. Wir definieren rekursiv die Relation $\sqsubseteq$ auf allen Teilmengen von U.

Für alle $U_1, U_2 \subseteq U$ gilt

$$\{\bullet\} \sqsubseteq U_1$$
$$U_1 \sqsubseteq U_1 \times U_2$$
$$U_2 \sqsubseteq U_1 \times U_2$$

Wenn $U_1 \sqsubseteq U_2$ dann nennen wir U_2 eine *Sortenerweiterung* von U_1. Wir definieren

kanonisch für jedes Element $u \in U_2$ die Projektion von u auf U_1

$$u|_{U_1} = \begin{cases} \bullet \text{ falls } U_1 = \{\bullet\} \\ u_1 \text{ falls } U_2 = U_1 \times U_3 \text{ und } u = (u_1, u_3) \\ u_1 \text{ falls } U_2 = U_3 \times U_1 \text{ und } u = (u_3, u_1) \\ u_1 \text{ falls } U_2 = U_3 \times U_1 \times U_4 \text{ und } u = (u_3, u_1, u_4) \end{cases}$$

Diese Definition erweitern wir kanonisch auf Multimengen, d.h. die Einschränkung einer Multimenge ist gerade die Multimenge der eingeschränkten Elemente.

Bei einer Datenerweiterung verlangen wir, daß die Netze gleich sind, die den beiden Systemen zugrunde liegen. Außerdem sind die Sorten der Stellen des erweiterten Netzes Sortenerweiterungen der Sorten der Stellen des Ausgangsnetzes. Wenn man die Kanteninschriften des erweiterten Netzes auf die Sorten des Ausgangsnetzes einschränkt, erhält man gerade die Kanteninschriften des Ausgangsnetzes. Die Aktivierungsbedingungen des erweiterten Netzes können stärker sein, als die Aktivierungbedingungen des Ausgangsnetzes. Dies sind starke syntaktische Restriktionen, durch die der Nachweis einer korrekten Datenerweiterung sehr vereinfacht wird.

Definition 5.13 (Datenerweiterung)

Sei $N = (P, T, F)$ ein Netz und seien weiterhin $N_1 = (N, \mathcal{A}, \mathcal{X}, d_1, w_1, g_1)$ und $N_2 = (N, \mathcal{A}, \mathcal{X}, d_2, w_2, g_2)$ algebraische Petrinetze, denen das gleiche Netz, die gleiche Algebra und die gleiche sortierte Variablenmenge zugrunde liegen. N_2 ist eine *Datenerweiterung* von N_1, wenn für alle Stellen $p \in P$ die Sorte $d_2(p)$ eine Sortenerweiterung der Sorte $d_1(p)$ ist und für jede Belegung β gilt:

(i) für die Aktivierungsfunktionen g_1 und g_2 gilt: $g_2(t)(\beta)$ impliziert $g_1(t)(\beta)$ für alle Transitionen $t \in T$

(ii) für alle Kanten $f \in F$ mit $f = (p, t)$ oder $f = (t, p)$ gilt: $w_1(f)(\beta) = (w_2(f)(\beta))|_{d_1(p)}$. ⋆

Wir betrachten zwei Systeme $\Sigma_1 = (N_1, M_1)$ und $\Sigma_2 = (N_2, M_2)$, so daß N_2 eine Datenerweiterung von N_1 ist. Sei $K = (B, E, \lessdot)$ ein Kausalnetz und r eine Σ_2-Beschriftung für K. Dann ist die Funktion r', die für alle $b \in B$ durch $r'(b) = (p, x|_{d_1(p)})$ falls $r(b) = (p, x)$ definiert ist, eine Σ_1-Beschriftung für K. Wir bezeichnen (K, r') auch als Einschränkung $(K, r)|_{d_1}$. Analog definieren wir die Einschränkung einer Markierung M von Σ_2 auf eine Markierung $M|_{d_1}$ von Σ_1.

Die Menge aller eingeschränkten verteilten Abläufe[1] von Σ_2 bezeichnen wir mit $\mathcal{R}(\Sigma_2)|_{d_1} = \{\rho\,|_{d_1}\,|\,\rho \in \mathcal{R}(\Sigma_2)\}$.

Bemerkung 5.14 Wenn $M_1 = M_2|_{d_1}$, dann ist jeder auf d_1 eingeschränkte verteilte Ablauf von Σ_2 ein Prozeß von Σ_1, aber nicht unbedingt ein Ablauf von Σ_1. ⋆

[1] Zur Erinnerung: Die Menge aller verteilten Abläufe von eines Systems Σ bezeichnen wir mit $\mathcal{R}(\Sigma)$.

Korrekte Datenerweiterung

Bei einer *korrekten Datenerweiterung* wird gefordert, daß jeder verteilte Ablauf des verfeinerten Systems durch Einschränkung der Beschriftung ein verteilter Ablauf des abstrakten Systems wird. Das abstrakte System kann aber mehr verteilte Abläufe als das verfeinerte System haben. Da eine temporale Formel in einem System genau dann gilt, wenn sie in jedem verteilten Ablauf des Systems gilt, bleiben bei einer korrekten Datenverfeinerung alle Eigenschaften des abstrakten Systems im verfeinerten System erhalten.

Definition 5.15 (Korrekte Datenerweiterung)

Seien $\Sigma_1 = (N_1, M_1)$ und $\Sigma_2 = (N_2, M_2)$ Systeme, so daß N_2 eine Datenerweiterung von N_1 ist. Sei weiter d_1 die Sortenfunktion von Σ_1 und sei $M_1 = M_2|_{d_1}$. Σ_2 ist eine *korrekte Datenerweiterung* von Σ_1,wenn $\mathcal{R}(\Sigma_2)|_{d_1} \subseteq \mathcal{R}(\Sigma_1)$. $\star$

Wenn bereits eine Datenerweiterung gegeben ist, muß sichergestellt werden, daß die Anfangszustände der Systeme übereinstimmen und daß durch zusätzliche Synchronisation die Progressannahme des abstrakten Petrinetzes nicht verletzt wird. Im nächsten Satz zeigen wir, daß das bereits eine hinreichende Bedingung für korrekte Datenerweiterung ist.

Satz 5.16 Sei $\Sigma_2 = (N_2, M_2)$ eine Datenerweiterung von $\Sigma_1 = (N_1, M_1)$. Das System Σ_2 ist genau dann eine korrekte Datenerweiterung von Σ_1, wenn $M_1 = M_2|_{d_1}$ und für jeden verteilten Ablauf (K, r) von Σ_2 gilt: in $r(\mathrm{K}^\circ)|_{d_1}$ ist keine Transition von Σ_1 aktiviert. $\star$

Beweis: Durch Bemerkung 5.14 wissen wir bereits, daß jeder eingeschränkte Ablauf von Σ_2 ein Prozeß von Σ_1 ist. Die zusätzliche Bedingung, daß in $r(\mathrm{K}^\circ)|_{d_1}$ keine Transition von Σ_1 aktiviert ist, garantiert, daß jeder dieser Prozesse auch wirklich ein Ablauf von Σ_1 ist. q.e.d.

Falls beide Systeme nur endliche Abläufe haben, reicht es sogar zu zeigen, daß für jede tote, erreichbare Markierung M von Σ_2 die Markierung $M|_{d_1}$ auch tot in Σ_1 ist.

Konservative Datenerweiterung

Bei einer *konservativen Datenerweiterung* werden neue Variablen eingeführt, die keinen Einfluß auf den Algorithmus haben. Wir erreichen dies durch starke syntaktische Restriktionen, so daß man direkt am Petrinetzmodell ablesen kann, ob die Einführung einer neuen Variablen eine konservative Datenerweiterung ist. Durch die syntaktischen Kriterien, ist eine konservative Datenerweiterung insbesondere unabhängig vom Anfangszustand der Systeme. Konservative Datenerweiterung wird deshalb nicht für Systeme, sondern für algebraische Petrinetze definiert.

Definition 5.17

Seien $N_1 = (N, \mathcal{A}, \mathcal{X}, d_1, w_1, g_1)$ und $N_2 = (N, \mathcal{A}, \mathcal{X}, d_2, w_2, g_2)$ algebraische Petrinetze, so daß N_2 eine Datenerweiterung von N_1 ist. N_2 ist eine *konservative Datenerweiterung* von N_1, wenn für jede Anfangsmarkierung M von N_2 gilt: $\mathcal{R}(N_2, M)|_{d_1} = \mathcal{R}(N_1, M|_{d_1})$. $\star$

Durch gleiche Variablen an eingehenden Kanten wird Synchronisation erreicht. Zusätzliche Synchronisation durch erweiterte Sorten ist bei konservativer Datenverfeinerung nicht zulässig. Ein notwendiges Kriterium für konservative Datenverfeinerung ist deshalb, daß an eingehenden Kanten einer Transition nur unterschiedliche Variablen für die "neuen" Sorten verwendet werden.

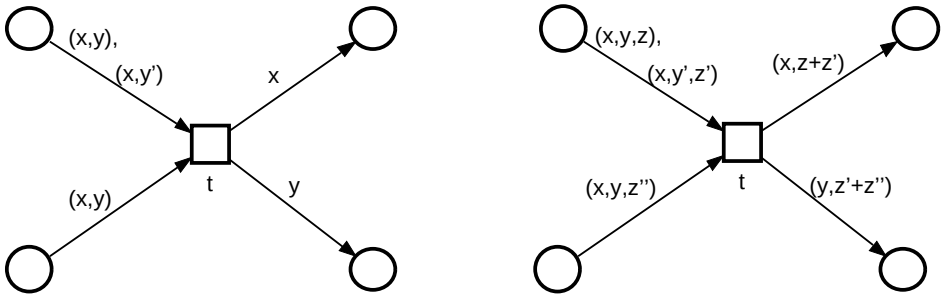


Abb. 5.11: Konservative Datenerweiterung an der Transition t

In Abb. 5.11 ist ein Beispiel für die Datenerweiterung einer Transition dargestellt, die dieses Kriterium erfüllt. Abb. 5.11 zeigt dagegen zwei Datenerweiterungen, die dieses Kriterium nicht erfüllen.

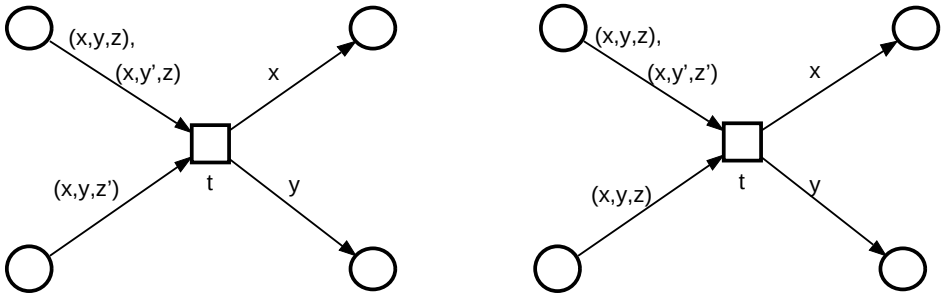


Abb. 5.12: Zwei Datenerweiterungen von Σ_1, die das Kriterium von Satz 5.18 nicht erfüllen

Im erweiterten Petrinetz dürfen bei einer konservativen Datenerweiterung auch keine zusätzlichen Aktivierungsfunktionen für zusätzliche Synchronisation sorgen. Wir fordern daher, daß die Aktivierungsfunktionen im abstrakten und im erweiterten Petrinetz gleich sind und nur über die "alten" Sorten Aussagen machen. Daß dies zusammen mit den obigen Kriterium auch hinreichend ist, wird im nächsten Satz bewiesen.

Satz 5.18 Sei das algebraische Petrinetz $N_2 = (N, \mathcal{A}, \mathcal{X}, d_2, w_2, g)$ eine Datenerweiterung von $N_1 = (N, \mathcal{A}, \mathcal{X}, d_1, w_1, g)$, mit derselben Aktivierungsfunktion g für beide Petrinetze. N_2 ist eine konservative Datenerweiterung von N_1, wenn

(i) für jede Transition t gilt: wenn die Variable x in $g(t)$ frei vorkommt, dann gibt es ein $p \in {}^\bullet t$, so daß x in $w_1(p,t)$ frei vorkommt (d.h. alle Aussagen der Aktivierungsbedingung beziehen sich auf "alte" Variablen),

(ii) für jede Transition t gilt: an allen eingehenden Kanten stehen nur Terme, der Form (x, n_i), wobei x die "alten" Variablen bezeichnet und n_i die "neuen". Zusätzlich sind alle n_i an eingehenden Kanten von t paarweise verschieden. $\star$

Beweis: Eine Transition t kann genau dann im Modus (t, β) in N_1 schalten, wenn sie im Modus (t, β) in N_2 schalten kann und $(t, \beta)^-$ in N_1 ist gleich $(t, \beta)^-|_{d_1}$ in N_2 und $(t, \beta)^+$ in N_1 ist gleich $(t, \beta)^+|_{d_1}$ in N_2. Daraus folgt $\mathcal{R}(N_2, M)|_{d_1} = \mathcal{R}(N_1, M|_{d_1})$ für jede Anfangsmarkierung M von N_2. q.e.d.

5.4 Klassische Verfeinerungen

Sequentielle versus verteilte Abläufe

Nicht in jedem Verfeinerungsschritt ist ein Verfeinerungsbegriff, der auf Halbordnungen beruht einsetzbar, da ein Vergleich der verteilten Abläufe von zwei Systemen entweder unmöglich oder nicht sinnvoll ist.

Wenn zum Beispiel neue (zusätzliche) Aktionen eingeführt werden, bleiben die verteilten Abläufe nicht erhalten. In vielen Verfeinerungsschritten wird nicht nur eine neue Variable oder eine neue Transition eingeführt, sondern das gesamte System wird transformiert. Die Abläufe des Ausgangssystems und des verfeinerten Systems sind dann nicht mehr direkt vergleichbar, sondern werden durch eine Funktion o.ä. aufeinander abgebildet. Meist wird dazu jeder globale Zustand (jede Markierung) des verfeinerten Systems auf einen globalen Zustand des Ausgangssystems abgebildet. Bei dieser globalen zustandsbasierten Betrachtungsweise ist es nicht sinnvoll, Nebenläufigkeit von Aktionen zu betrachten.

Wir führen nun einen Verfeinerungsbegriff ein, der auf den sequentiellen Abläufen der Systeme basiert. Wir orientieren uns am Verfeinerungsbegriff von Abadi und Lamport [AL91]. Abadi und Lamport definieren in [AL91] einen Verfeinerungsbegriff für *Zustandsmaschinen*. In einer Zustandsmaschine gibt es sichtbare und unsichtbare Variablen und damit auch sichtbare und unsichtbare Aktionen. Wir übertragen diesen Verfeinerungsbegriff auf algebraische Petrinetze. Dieser Begriff ist sehr allgemein. Durch Einschränkungen kann man leichter zu handhabende Begriffe definieren, z.B. Superposition [CM88, BS96] oder Datenverfeinerung [dRE98]. Durch die Definition eines eingeschränkten Verfeinerungsbegriffs kann man einfachere Beweiskriterien für die Korrektheit eines solchen Schrittes angeben.

Da wir uns in dieser Arbeit aber auf Verfeinerungen mit Halbordnungen konzentrieren, begnügen wir uns mit dem allgemeinen Begriff, der in den meisten Fällen, in denen halbordnungsbasierte Verfeinerung nicht möglich oder ungünstig ist, ausreicht.

Beobachterverfeinerung

Bei einer Datenerweiterung müssen dem Ausgangsnetz und dem erweiterten Netz dieselben Netze zugrunde liegen. Dies ist eine starke Restriktion. In der nun zu definierenden Beobachterverfeinerung kann das verfeinerte System eine völlig andere Struktur als das Ausgangssystem haben.

Wir definieren für jedes System einen Beobachter, der jede Markierung des Systems auf einen *beobachtbaren Zustand* abbildet. Beide an einem Verfeinerungsschritt beteiligten Systeme haben die gleiche Menge beobachtbarer Zustände. Diese beobachtbaren Zustände müssen wiederum nichts mit der Netzstruktur zu tun haben. Da wir Eigenschaften von Systemen erhalten wollen, wählen die Beobachter immer so, daß sie genug sehen, um zu entscheiden, ob eine bestimmte Eigenschaft erhalten bleibt.

Zum Beispiel kommen in den Formeln, mit denen wir die Korrektheit eines Systems spezifizieren, meist nicht alle Stellen des Systems vor. Es genügt also, wenn ein Beobachter des Systems nur die in einer Formel vorkommenden Stellen sieht, um zu entscheiden, ob sich das System korrekt verhält.

Beispiel: Erreichbarkeit im Graphen

Wir erklären Beobachterverfeinerung zuerst anhand eines Beispiels. Sei $G = (A, N)$ ein endlicher Graph mit der Knotenmenge A und der Kantenmenge $N \subseteq A \times A$. Sei i ein ausgezeichnetes Element von A. Wir möchten wissen, welche Knoten von i aus erreichbar sind. Ein (nicht nachrichtenbasierter) Algorithmus, der dieses Problem löst, ist in Abb. 5.13 modelliert.

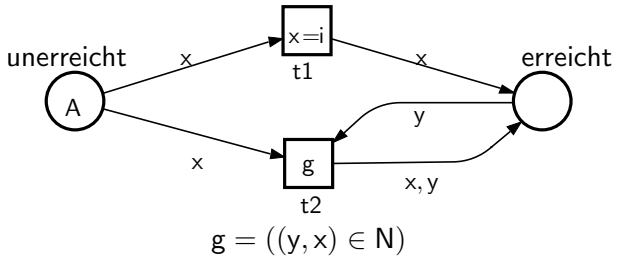


Abb. 5.13: Σ_g: Erreichbarkeit in einem Graph (nicht nachrichtenbasiert)

Zu Beginn sind alle Knoten unerreicht. Der Knoten i geht zuerst in den Zustand erreicht über. Wenn ein Knoten b bereichts erreicht ist und der Knoten a unerreicht ist und im Graphen eine Kante von b nach a geht, dann kann die Transition im Modus $[x = a, y = b]$ schalten. Dabei geht a in den Zustand erreicht über, b bleibt in diesem Zustand.

Dieser Algorithmus terminiert für jeden endlichen Graphen und im Endzustand liegen auf der Stelle erreicht genau die Knoten, die von i aus erreichbar sind. Alle anderen Knoten liegen auf der Stelle unerreicht.

Wir verfeinern diesen Algorithmus nun zu einem nachrichtenbasierten Algorithmus. Wir betrachten die Elemente der Menge A als Agenten und die Elemente von N als unidirektionale Kommunikationskanäle. Ein Agent x kann einem Agenten y genau dann eine Nachricht schicken, wenn $(x, y) \in N$.

Für jeden Agenten x bezeichnet $M(x) = \{(y, x) \mid (x, y) \in N\}$ eine Menge von Nachrichten von x und zwar an jeden Nachbarn von x eine Nachricht. Wieder sind anfangs alle Agenten unerreicht. Zuerst geht i in den Zustand erreicht über und schickt jedem Nachbar eine Nachricht. Jeder unerreichte Agent, der eine Nachricht

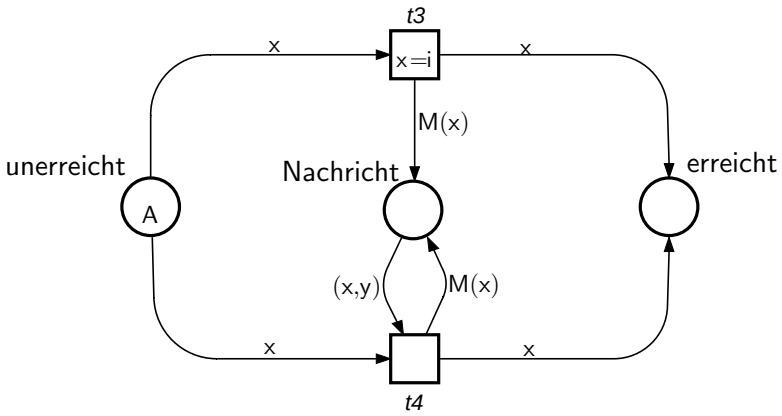


Abb. 5.14: Σ'_g: Erreichbarkeit in einem Graph (nachrichtenbasiert)

erhält, geht in den Zustand erreicht über und schickt eine Nachricht an alle seine Nachbarn.

Wir definieren nun Beobachter, so daß Σ'_g eine Beobachterverfeinerung von Σ_g ist. Der Beobachter Obs von Σ_g sieht beide Stellen unerreicht und erreicht. Der Beobachter Obs$'$ von Σ'_g sieht ebenfalls die Stellen unerreicht und erreicht. Die Stelle Nachricht bleibt ihm verborgen.

Die Menge der beobachtbaren Zustände für beide Systeme ist $2^A \times 2^A$, die Menge aller geordneten Paare aus Teilmengen von A. Mit Hilfe der folgenden Funktionen kann jede Markierung M von Σ_g bzw. M' von Σ'_g aus einen beobachtbaren Zustand abgebildet werden.

$$\begin{aligned} \text{obs}(M) &= (\mathsf{erreicht}, \mathsf{unerreicht}) \\ \text{obs}'(M') &= (\mathsf{erreicht}, \mathsf{unerreicht}) \end{aligned}$$

Beide Beobachter beobachten dieselben sequentiellen Abläufe, deshalb ist Σ'_g eine Beobachterverfeinerung von Σ_g bzgl. Obs und Obs$'$. Die Systeme arbeiten korrekt, wenn irgendwann für immer alle von i aus erreichbaren Agenten im Zustand erreicht sind. Wir brauchen nur zu beweisen, daß Σ_g korrekt ist. Da beide Beobachter die Stelle erreicht sehen, können wir die Korrektheit von Σ'_g aus der Korrektheit von Σ_g folgern.

Ein Beobachter sieht ein System sozusagen durch eine Brille, die Teile des System ausblendet. Da ein Beobachter immer nur ganze Markierungen (also globale Zustände) des Systems auswertet, ist es nicht sinnvoll, verteilte Abläufe zu betrachten. Wir definieren einen Beobachter auf der Grundlage von sequentiellen Abläufen eines Systems.

In Definition 4.2 haben wir sequentielle Abläufe als alternierende Folgen von Markierungen und Aktionen eingeführt. In diesem Abschnitt geben wir die Aktionen nicht an, sondern stellen einen sequentiellen Ablauf einfach als Folge von Markierungen dar. Wenn jede Aktion (t, β) des Systems eindeutig durch $(t, \beta)^+$ und$(t, \beta)^-$ bestimmt ist, dann gehen die nicht angegebenen Aktionen sogar eindeutig aus der

Folge von Markierungen hervor. In der Petrinetztheorie nennt man solche Petrinetze *transitionsschlicht*.

Definition 5.19 (Beobachter, beobachtbare Markierung)

Sei $\Sigma = (N, M_0)$ ein System und sei $\mathcal{M}$ die Menge aller Markierungen von Σ. Ein *Beobachter* $\mathrm{Obs} = (\mathcal{O}, \mathrm{obs})$ von Σ besteht aus einer Menge $\mathcal{O}$ und einer Funktion $\mathrm{obs} : \mathcal{M} \longrightarrow \mathcal{O}$. Wir nennen $\mathcal{O}$ die Menge der beobachtbaren Zustände und $\mathrm{obs}(M)$ die *beobachtbare Markierung* von M bzgl. obs für jede Markierung M von Σ. $\star$

Ein System kann Aktionen ausführen, ohne daß die beobachtbare Markierung verändert wird. Der Beobachter bemerkt solche "Stotterschritte" nicht. Er sieht nicht, daß überhaupt eine Aktion ausgeführt wurde. Bevor wir beobachtbare Abläufe definieren, benötigen wir noch den Begriff der "Stotterfreiheit".

Sei $\rho = (x_0, x_1, x_2 \dots)$ eine beliebige Folge. Wenn $x_i = x_{i+1}$ ist, dann nennen wir (x_i, x_{i+1}) einen Stotterschritt dieser Folge. Die *stotterfreie Version* dieser Folge erhalten wir, indem wir jede (endliche oder unendliche) Teilfolge $(x_i, x_{i+1}, \dots)$ mit $x_i = x_{i+1} = \dots$ durch das Folgenglied x_i ersetzen. Wir bezeichnen die stotterfreie Version einer Folge ρ mit $\natural\rho$. Die formale Definition der stotterfreien Version einer Folge ist im Anhang A.3 angegeben.

Definition 5.20 (beobachtbarer Ablauf)

Sei $\Sigma = (N, M_0)$ ein System und $\mathrm{Obs} = (\mathcal{O}, \mathrm{obs})$ ein Beobachter von Σ. Für jeden sequentiellen Ablauf $\rho = (M_0, M_1, M_2 \dots)$ von Σ definieren wir den *beobachtbaren Ablauf* bzgl. obs als stotterfreie Folge der beobachtbaren Markierungen $\mathrm{obs}(\rho) = \natural(\mathrm{obs}(M_0), \mathrm{obs}(M_1), \mathrm{obs}(M_2), \dots)$. Die Menge aller beobachtbaren Abläufe von Σ bzgl. des Beobachters Obs bezeichnen wir mit $\mathcal{R}_{\mathrm{Obs}}(\Sigma)$. $\star$

Ein konkretes System ist eine Verfeinerung eines abstrakten Systems bzgl. bestimmter Beobachter der Systeme, wenn der abstrakte Beobachter im abstrakten System jeden Ablauf beobachten kann, den der konkrete Beobachter im konkreten System beobachtet.

Definition 5.21 (Beobachterverfeinerung)

Seien Σ_1 und Σ_2 Systeme und seien $\mathrm{Obs}_1 = (\mathcal{O}_1, \mathrm{obs}_1)$ und $\mathrm{Obs}_2 = (\mathcal{O}_2, \mathrm{obs}_2)$ Beobachter von Σ_1 bzw. Σ_2. Σ_2 ist eine *Beobachterverfeinerung* von Σ_1 bzgl. Obs_1 und Obs_2, wenn $\mathcal{O}_1 = \mathcal{O}_2$ und $\mathcal{R}_{\mathrm{Obs}_2}(\Sigma_2) \subseteq \mathcal{R}_{\mathrm{Obs}_1}(\Sigma_1)$.

Die Systeme heißen *beobachteräquivalent* bzgl. Obs_1 und Obs_2, wenn $\mathcal{R}_{\mathrm{Obs}_2}(\Sigma_2) = \mathcal{R}_{\mathrm{Obs}_1}(\Sigma_1)$. $\star$

Welche Eigenschaften bei einer Beobachterverfeinerung erhalten bleiben, hängt natürlich stark von den Beobachtern ab. Wir diskutieren in dieser Arbeit nicht ausführlich, wie man beweist, daß ein System eine Beobachterverfeinerung eines anderen Systems ist. Eine allgemeine, klassische Beweismethode für Verfeinerungen ist Simulation. Dabei zeigt man, daß das Ausgangssystem das verfeinerte System simuliert, also daß das Ausgangssystem dem verfeinerten System alles nachmachen kann, so daß die beobachtbaren Markierungen immer gleich bleiben. Wir geben hier nur ein Simulationskriterium für terminierende Systeme an.

Satz 5.22 Seien $\Sigma_1 = (N_1, M_1^0)$ und $\Sigma_2 = (N_2, M_2^0)$ Systeme und seien Obs_1 und Obs_2 Beobachter von Σ_1 bzw. Σ_2. Wir nehmen an, daß beide Systeme terminieren, d.h. alle sequentiellen Abläufe von Σ_1 und Σ_2 sind endlich. Σ_2 ist eine Beobachterverfeinerung von (Σ_1, M_1^0) bzgl. Obs_1 und Obs_2, wenn die folgenden Bedingungen gelten:

(i) $\text{obs}_1(M_1^0) = \text{obs}_2(M_2^0)$ (beide Beobachter sehen die gleichen Anfangszustände),

(ii) sei M_2 eine erreichbare Markierung von Σ_2 und sei $M_2 \stackrel{(t,\beta)}{\rightarrow} M_2'$ ein Schritt, dann gilt

  (a) $\text{obs}_2(M_2) = \text{obs}_2(M_2')$ oder

  (b) für jede erreichbare Markierung M_1 von Σ_1 mit $\text{obs}_1(M_1) = \text{obs}_2(M_2)$ gibt es eine Markierung M_1' von Σ_1 mit $\text{obs}_1(M_1') = \text{obs}_2(M_2')$ und eine Aktion (t', β'), so daß $M_1 \stackrel{(t',\beta')}{\rightarrow} M_1'$ ein Schritt ist,

(iii) wenn M_2 eine tote Markierung von Σ_2 ist und M_1 eine erreichbare Markierung von Σ_1 mit $\text{obs}_1(M_1) = \text{obs}_2(M_2)$ ist, dann ist M_1 auch tot in Σ_1. $\star$

Beweis: Wir müssen zeigen, daß jeder beobachtbare Ablauf von Σ_2 auch ein beobachtbarer Ablauf von Σ_1 ist. Sei ρ ein Ablauf von Σ_2. Mit den Bedingungen (i)-(ii) können wir einen Ablauf ρ' von Σ_1 konstruieren, so daß $\text{obs}_1(\rho_1) = \text{obs}_2(\rho_2)$: Der Anfangszustand von ρ entspricht dem Anfangszustand von Σ_1. Immer wenn in ρ eine Aktion schaltet, die die beobachtbare Markierung nicht verändert (Fall (ii)a), dann wird das Anfangsstück von ρ' nicht fortgesetzt. Im Fall (ii)b wird das Anfangsstück von ρ' durch den Schritt $M_1 \stackrel{(t',\beta')}{\rightarrow} M_1'$ fortgesetzt. q.e.d.

Wir betrachten noch einmal unser Beispiel. Die beobachtbaren Anfangsmarkierungen stimmen überein. Beide Systeme haben nur endliche Abläufe, denn es gibt anfangs nur endlich viele Marken auf der Stelle unerreicht und jedesmal wenn eine Transition schaltet, wird eine Marke von dieser Stelle entfernt. Zum Beweis der Bedingungen (ii) und (iv) benötigen wir folgende leicht überprüfbare Invarianten von Σ_g':

$$\Sigma_g' \models \Box\, \mathsf{Nachricht}(x, y) \rightarrow \mathsf{erreicht}(y)$$
$$\Sigma_g' \models \Box(\mathsf{unerreicht}(x) \wedge \mathsf{erreicht}(y) \wedge (x, y) \in N) \rightarrow \mathsf{Nachricht}(x, y)$$

Nicht jede Beobachterverfeinerung läßt sich mit Simulation beweisen. Diskussionen über Simulation als Beweismethode und über andere Beweismethoden, z.B. Rückwärtssimulation findet man reichlich in der Literatur (siehe z.B. [AL91]). Zum Beweis der in unserer Herleitung des GHS-Algorithmus vorkommenden Beobachterverfeinerung reicht Simulation aus.

Sei Σ ein System mit der Stellenmenge P. Sei weiter $\mathcal{M}$ die Menge aller Markierungen von Σ. Der Beobachter $\text{Obs}_{id} = (\mathcal{M}, id_{\mathcal{M}})$ kann alle sequentiellen Abläufe von Σ beobachten, d.h. die Menge der beobachtbaren Abläufe ist gleich der Menge der sequentiellen Abläufe des Systems. Von besonderer Bedeutung sind auch Beobachter, die einige Stellen des Netzes sehen und andere Stellen nicht. Sei $S \subseteq P$, dann

definieren wir $\mathrm{Obs}_S = (\mathcal{M}|_S, id|_S)$ als den Beobachter, der von jeder Markierung genau die Stellen aus S sieht.

Wir werden ausschließlich Beobachterverfeinerungen betrachten, in denen der Beobachter des abstrakten Systems eine der beiden oben beschriebenen Formen hat. Wenn man dann noch den konkreten Beobachter geschickt wählt, kann man alle temporallogischen Aussagen über das abstrakte System bzw. alle temporallogischen Aussagen über das abstrakte System, in denen nur Stellensymbole aus S vorkommen, in Aussagen über das konkrete System übersetzen und zeigen, daß deren Gültigkeit erhalten bleibt.

Teil III

Der Algorithmus von Gallager, Humblet und Spira

In diesem Teil der Arbeit verwenden wir die in Teil II vorgestellten Verfeinerungstechniken und Ergebnisse zur Modellierung und zum Beweis des Algorithmus von Gallager, Humblet und Spira (kurz: GHS-Algorithmus, [GHS83]). Der GHS-Algorithmus ist ein nachrichtenbasierter Algorithmus zur Berechnung des minimalen spannenden Baumes in einem gewichteten Kommunikationsnetzwerk.

Wegen seiner Komplexität hat sich dieser Algorithmus zu einem Prüfstein für Verifikationsmethoden verteilter Algorithmen entwickelt. Die Korrektheit des Algorithmus wurde schon mehrfach nachgewiesen [CG88, Hes99a, SdR87, WLL88], aber die Beweise sind meist schwer nachvollziehbar oder unvollständig. Wir diskutieren die anderen Beweise im Anschluß an unsere Darstellung.

6 Problemstellung und Vorbetrachtungen

Wir beschreiben zuerst das Problem, das der GHS-Algorithmus löst und die mathematische Idee, auf der die Lösung basiert.

6.1 Freundinnentarif

Auf der ganzen Welt verteilt gibt es wißbegierige Frauen. Jede wißbegierige Frau hat Freundinnen, die natürlich auch alle wißbegierig sind. Wenn eine wißbegierige Frau wieder mal eine Neuigkeit erfährt, ruft sie sofort alle ihre Freundinnen an, um ihnen die Neuigkeit zu erzählen, die dann wiederum ihre Freundinnen anrufen usw.

Jede wißbegierige Frau kennt nur ihre Freundinnen und keine anderen wißbegierigen Frauen. Die Freundschaften unter den wißbegierigen Frauen sind so verteilt, daß jede Neuigkeit, die sich von einer beliebigen wißbegierigen Frau ausgehend verbreitet, jede wißbegierige Frau erreicht.

Die Telefonrechnungen aller wißbegierigen Frauen werden aus einem von anonymen Gönnern gesponsorten Fond bezahlt. Sei es nun, daß das Interesse an wißbegierigen Frauen abnimmt, oder daß es zu viele Neuigkeiten gibt, jedenfalls ist die Gesamttelefonrechnung zu hoch.

Wo kann nun gespart werden, wenn gewährleistet sein soll, daß jede wißbegierige Frau jede Neuigkeit erfährt? Wenn zum Beispiel zwei deutsche Freundinnen eine gemeinsame Freundin in China haben, gibt es bei der Verbreitung einer Neuigkeit immer zwei Telefonate zwischen China und Deutschland. Wenn eine der deutschen Frauen und die chinesische Frau ihre Freundschaft beenden, werden immer noch alle drei informiert und die Gesamttelefonrechnung wird geringer.

Durch die Kündigung von Freundschaften eröffnet sich den wißbegierigen Frauen eine Einsparungsmöglichkeit, die zwar hart für sie ist, aber es geht ja schließlich um Geld: Zu jeder Freundschaft gibt es einen Durchschnittswert für den Preis eines Telefonats zwischen den beiden Freundinnen, den jede der beiden Freundinnen kennt. Nun sollen die wißbegierigen Frauen untereinander so viele Freundschaften kündigen, daß jede wißbegierige Frau jede Neuigkeit erfährt und die Gesamttelefonrechnung minimal ist.

Es ist also ein Verfahren gesucht, in dem jede Frau nur mit ihren Freundinnen telefoniert und zum Schluß weiß, welche Freundschaften sie kündigen muß und welche Freundschaften notwendig zur preiswerten Neuigkeitsverbreitung bleiben.

6.2 Sprechweise

Im folgenden werden wir nicht über Freundinnen und Freundschaften reden, sondern über Agenten und Kommunikationskanäle. Wir betrachten ein zusammenhängendes, gewichtetes Kommunikationsnetzwerk (A, N, w). Dabei sind A eine Menge von Agenten, N eine Menge bidirektionaler Kommunikationskanäle und $w : N \longrightarrow \mathbb{R}$ eine Gewichtungsfunktion, die eindeutig[1] jedem Kommunikationskanal ein Gewicht (üblicherweise eine reelle Zahl) zuordnet.

Ein *spannender Baum* des Kommunikationsnetzwerks ist ein azyklischer, zusammenhängender Teilgraph des Graphen, der dem Netzwerk zugrunde liegt (siehe Abschnitt A.2). Ein *minimaler spannender Baum* ist ein spannender Baum, dessen Summe der Kantengewichte gegenüber allen anderen spannenden Bäumen minimal ist.

Zwei Agenten, die durch einen Kommunikationskanal miteinander verbunden sind, heißen *Nachbarn*. Zwei Agenten, die durch einen Kommunikationskanal miteinander verbunden sind, der zum minimalen spannenden Baum des Netzwerks gehört, heißen *Baumnachbarn*. Jeder Agent kennt seine Nachbarn und die Gewichte der Kommunikationskanäle zu seinen Nachbarn.

Der GHS-Algorithmus ist ein nachrichtenbasierter Algorithmus auf einem zusammenhängenden, eindeutig gewichteten Kommunikationsnetzwerk, der den minimalen spannenden Baum des Netzwerkes berechnet, so daß jeder Agent nach der Berechnung weiß, welche seiner Nachbarn Baumnachbarn sind.

6.3 Die mathematische Idee des GHS-Algorithmus

In diesem Abschnitt formulieren wir die mathematische Idee, die dem GHS-Algorithmus zugrunde liegt, als graphentheoretischen Satz.

Bei der Problemstellung in Abschnitt 6.2 haben wir gefordert, daß die Gewichtungsfunktion eindeutig ist. Durch diese Einschränkung können wir das folgende graphentheoretische Resultat benutzen:

Satz 6.1 Zu jedem ungerichteten, zusammenhängenden, eindeutig gewichteten Graphen gibt es genau einen minimalen spannenden Baum. ⋆

Dieser Satz wird z.B. in [GHS83] bewiesen. Wir geben den einfachen Beweis an, da er beim Verständnis des GHS-Algorithmus hilft.

Beweis: Die Existenz eines spannenden Baumes ist klar. (Man entferne nur solange Kanten aus Kreisen, bis der Graph azyklisch ist.) Damit gibt es auch mindestens einen minimalen spannenden Baum. Angenommen es gibt zwei minmale spannende Bäume T_1 und T_2. Sei nun e_1 die Kante mit dem kleinsten Gewicht, die nur in einem der beiden Bäume vorkommt. O.B.d.A. sei e_1 aus T_1. Wenn man e_1 zu T_2

[1] Zur Erinnerung: ein bidirektionaler Kommunikationskanal zwischen den Agenten x und y wird durch die beiden geordneten Paare (x, y) und (y, x) repräsentiert. Die Gewichtungsfunktion ist eindeutig, wenn verschiedene Kanten verschiedene Gewichte haben, aber $w(x, y) = w(y, x)$ für alle Agenten gilt.

dazunimmt, entsteht ein Kreis, der mindestens eine Kante e_2 enthält, die nicht aus T_1 ist, denn in T_1 gibt es keine Kreise. Wegen der Wahl von e_1 und der Eindeutigkeit der Gewichtungsfunktion ist $w(e_1) < w(e_2)$. Wir definieren nun T_3 als den Baum, den man erhält, indem man in T_2 die Kante e_2 durch e_1 ersetzt. Die Summe der Kantengewichte von T_3 ist kleiner als die von T_2. Das ist ein Widerspruch dazu, daß T_2 ein minimaler spannender Baum ist. q.e.d.

Definition 6.2 (Fragment)

Sei (A, N, w) ein ungerichteter, zusammenhängender, eindeutig gewichteter Graph. Ein *Fragment* ist ein Teilbaum des (eindeutig bestimmten) minimalen spannenden Baumes von (A, N, w). Die *kleinste aus einem Fragment herausführende Kante* ist die Kante mit dem kleinsten Gewicht, die einen Agenten des Fragments mit einem Agent außerhalb des Fragments verbindet. ⋆

Der minimale spannende Baum ist selbst ein Fragment und außerdem das einzige Fragment, das keine aus dem Fragment herausführende Kante hat.

Der folgende Satz [GHS83] ist die mathematische Grundlage des GHS-Algorithmus.

Satz 6.3 Wir betrachten einen ungerichteten, zusammenhängenden, eindeutig gewichteten Graphen und ein Fragment des Graphen. Die kleinste aus dem Fragment herausführenden Kante gehört zum minimalen spannenden Baum des Graphen. ⋆

Auch hier geben wir den einfachen Beweis an, da er beim Verständnis des GHS-Algorithmus hilft.

Beweis: Angenommen, die kleinste aus einem Fragment F herausführende Kante e gehört nicht zum minimalen spannenden Baum T des Graphen. Wenn man e zu T hinzufügt, entsteht ein Kreis. In diesem Kreis gibt es eine Kante e', die aus F herausführt (siehe Abb. 6.1).

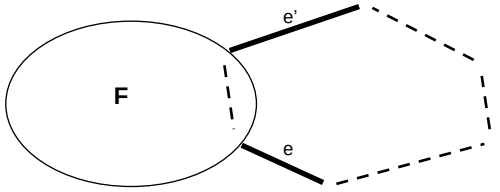


Abb. 6.1: Beweisillustration

Da e die kleinste aus F herausführende Kante ist, ist $w(e) < w(e')$. Man erhält einen spannenden Baum T', wenn man in T die Kante e' durch die Kante e ersetzt. Die Summe der Kantengewichte von T' ist kleiner als die von T. Das ist ein Widerspruch dazu, daß T der minimale spannende Baum ist. q.e.d.

Folgerung 6.4 Wenn zwei Fragmente durch eine Kante miteinander verbunden sind, die für mindestens eines der beiden Fragmente die kleinste aus dem Fragment herausführende Kante ist, dann ist die Vereinigung der beiden Fragmente zusammen mit der verbindenden Kante wieder ein Fragment. ⋆

6.4 Ein Beispiel

Wir stellen uns vor, daß ein eindeutig gewichtetes Kommunikationsnetzwerk auf einem Blatt Papier vor uns liegt. Unsere Aufgabe ist es, den minimalen spannenden Baum des Netzwerks zu bestimmen. Als Beispiel betrachten wir das Netzwerk in Abb. 6.2. Die Buchstaben bezeichnen die Namen der Agenten und die Zahlen an den Kanten geben Gewichte der Kanten an. Ein Fragment, das nur aus einem Knoten besteht, nennen wir *triviales* Fragment.

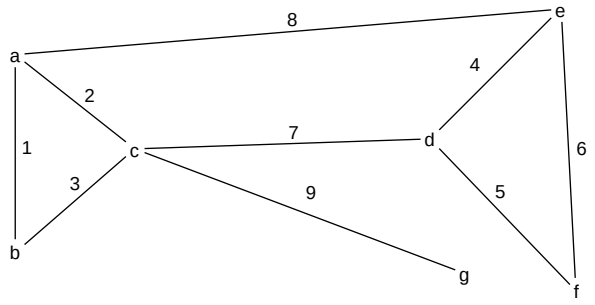


Abb. 6.2: Ein gewichtetes Kommunikationsnetzwerk

Satz 6.3 legt folgendes Vorgehen nahe: Wir beginnen mit den trivialen Fragmenten. Wenn zwei Fragmente durch eine Kante verbunden sind, die für mindestens eins der Fragmente die kleinste herausführende Kante ist, dann verbinden wir die beiden Fragmente zu einem neuen Fragment. Dieses neue Fragment besteht aus allen Knoten und Kanten der beiden Ausgangsfragmente und der verbindenden Kante. Wir führen diesen Schritt so oft aus, bis nur noch ein Fragment übrig ist. Dieses Fragment ist nach Satz 6.1 der minimale spannende Baum. Wir nennen diese Vorgehensweise Algorithmus 1.

In unserem Beispiel beginnen wir also mit den trivialen Fragmenten $(\{a\},\emptyset)$, $(\{b\},\emptyset)$, $(\{c\},\emptyset)$, $(\{d\},\emptyset)$, $(\{e\},\emptyset)$, $(\{f\},\emptyset)$, $(\{g\},\emptyset)$.

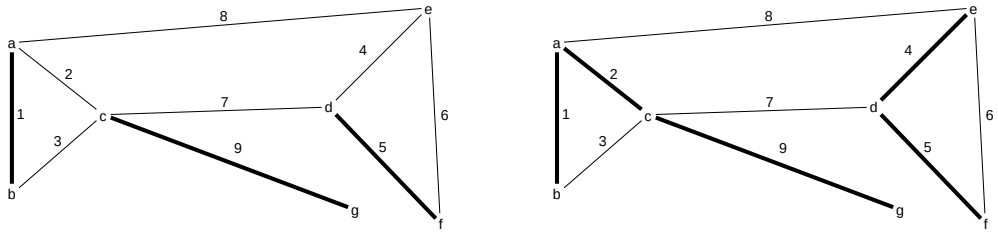


Abb. 6.3: Nach 3 Schritten und nach 5 Schritten

Wir verbinden a und b und die Kante mit dem Gewicht 1 zu einem neuen Fragment. Dasselbe machen wir mit c und g und der Kante 9 und mit d und f und der Kante 5. Danach haben wir noch 4 Fragmente mit den Agentenmengen $\{a,b\}$, $\{c,g\}$,$\{d,f\}$ bzw. $\{e\}$ (siehe Abb. 6.3, links).

Nun verbinden wir das Fragment aus den Agenten a und b mit dem Fragment aus den Agenten c und g über die Kante 2. Außerdem verbinden wir das Fragment aus d und f mit dem Fragment aus e über die Kante 4 (siehe Abb. 6.3, rechts). Wir haben nun

nur noch zwei Fragmente und die kleinste aus dem Fragment herausführende Kante ist für beide Fragmente die Kante mit dem Gewicht 7. Im letzten Schritt verbinden wir also noch diese beiden Fragmente und erhalten den in Abb. 6.4 angegebenen minimalen spannenden Baum.

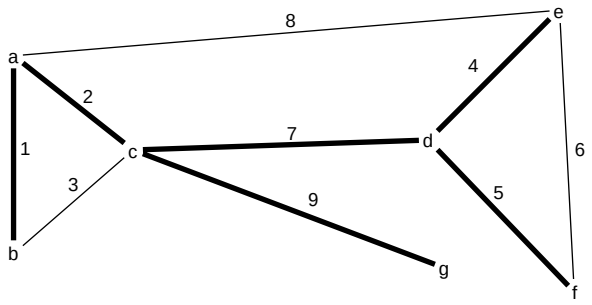


Abb. 6.4: Fett: der minimale spannende Baum

Beim Bestimmen des minimalen spannenden Baumes nach diesem Algorithmus, haben wir die Rolle einer "zentralen Kontrolle" übernommen, die das gesamte Netzwerk von oben überblicken kann. Wir haben die beschränkten Kommunikationsmöglichkeiten der Agenten völlig außer acht gelassen. Dieser Algorithmus ist nicht nachrichtenbasiert. Wir benutzen ihn aber als Ausgangspunkt unserer Verfeinerungshierarchie.

7 Die Herleitung des GHS-Algorithmus

7.1 Das initiale Modell

Informelle Beschreibung

Wir haben Algorithmus 1 informell bereits in Abschnitt 6.4 beschrieben. Der Algorithmus beginnt mit den trivialen Fragmenten. Wenn zwei Fragmente durch eine Kante verbunden sind, die für mindestens eins der Fragmente die kleinste herausführende Kante ist, dann verbinden wir die beiden Fragmente zu einem neuen Fragment. Der Algorithmus ist beendet, wenn es nur noch ein Fragment gibt.

Notationen

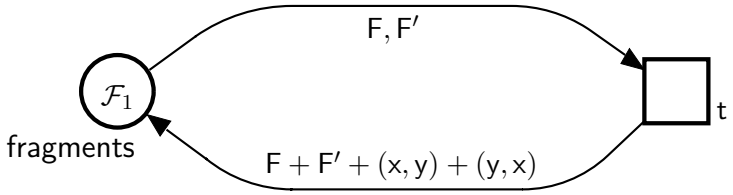
Bevor wir das Petrinetzmodell von Algorithmus 1 erklären, führen wir noch einige Notationen ein, die wir in der gesamten Herleitung der GHS-Algorithmus beibehalten. Wir betrachten ein beliebiges zusammenhängendes, eindeutig gewichtetes Kommunikationsnetzwerk (A, N, w), das aus mindestens zwei Agenten und mindestens einem Kommunikationskanal besteht. Nach Satz 6.1 hat dieses Netzwerk einen minimalen spannenden Baum, den wir mit $\mathrm{mst}(A, N, w)$ bezeichnen.

Die Sorte $\mathbb{F}\mathrm{ragment} = 2^A \times 2^N$ bezeichnet die Menge aller geordneten Paare, die aus einer Menge von Agenten und einer Menge von Kanten der Netzwerks bestehen. Im letzten Kapitel haben wir ein Fragment als Teilbaum des minimalen spannenden Baumes definiert. Jedes Fragment ist von der Sorte $\mathbb{F}\mathrm{ragment}$. Die Sorte $\mathbb{F}\mathrm{ragment}$ enthält beliebige Teilstrukturen des Kommunikationsnetzwerks, also auch Teilstrukturen, die keine Fragmente sind. Für ein Element F der Sorte $\mathbb{F}\mathrm{ragment}$ bezeichnet $F_{(1)}$ die erste Komponente, also die Menge der Agenten von F, und $F_{(2)}$ bezeichnet die zweite Komponente, also die Menge der Kanten von F. Wenn es klar ist, ob es sich um einen Agenten oder eine Kante handelt, schreiben wir auch $x \in F$ statt $x \in F_{(1)}$ und $(x, y) \in F$ statt $(x, y) \in F_{(2)}$. Für jeden Agenten $x \in A$ bezeichnet $N(x) = \{(x, y) \mid (x, y) \in N\}$ die Menge aller von x ausgehenden Kanten. Gemäß unserer Konvention aus Abschnitt A.2 bezeichnet $\overline{N(x)} = \{(y, x) \mid (x, y) \in N\}$ die Menge aller zu x eingehenden Kanten. Für jedes $F \in \mathbb{F}\mathrm{ragment}$ bezeichnet $N(F) = \bigcup_{x \in F_{(1)}} N(x)$ die Menge aller Kanten, die von Agenten von F ausgehen. $\overline{N(F)}$ ist entsprechend definiert. Mit $\mathrm{mo}(F)$ bezeichnen wir die kleinste aus F herausführende Kante, also $\mathrm{mo}(F) \in N(F)$, so daß $w(\mathrm{mo}(F)) = \min\{w(x, y) \mid (x, y) \in N(F) \wedge y \notin F\}$. Wenn F alle Agenten des Netzwerks enthält, dann ist $\mathrm{mo}(F)$ nicht definiert.

Die Addition zweier Fragmente[1] $F = (V, E)$ und $F' = (V', E')$, definieren wir als Vereinigung der Mengen der Agenten und der Mengen der Kanten: $F + F' = (V \cup V', E \cup E')$. Die Addition $F + (x, y)$ eines Fragments und einer Kante bedeutet, daß (x, y) zur Menge der Kanten des Fragments hinzugefügt wird, während die Menge der Agenten unverändert bleibt. $F + (x, y)$ ist also ein Schreibabkürzung für $F + (\emptyset, \{(x, y)\})$

Das Petrinetzmodell

Algorithmus 1 ist in Abb 7.1 als Petrinetzmodell Σ_1 dargestellt. Die Anfangsmarkierung ist gerade die Menge der trivialen Fragmente.



Sorten		Konstante
F, F′ :	𝔽ragment	$\mathcal{F}_1 = A \times \{\emptyset\}$
x, y :	A	
fragments :	𝔽ragment	

Aktivierungsbedingung

t : $x \in F \wedge y \in F' \wedge (y, x) = \mathrm{mo}(F')$

Abb. 7.1: Das initiale Modell Σ_1

Die Transition t ist aktiviert, wenn zwei Fragmente F und F' von der Stelle fragments die Aktivierungsbedingung $g(\mathsf{t})$ erfüllen. Mit der Aktivierungsbedingung formalisieren wir die Forderung, daß es eine verbindende Kante zwischen den zwei Fragmenten gibt, die für mindestens eins der Fragmente (in diesem Fall des Fragments F') die kleinste herausführende Kante ist. Durch Schalten von t verbinden sich F und F' zu einem neuen Fragment $F + F' + (x, y) + (y, x)$. Die Abbildungen 7.2 und 7.3 zeigen verteilte Abläufe von Σ_1 für unser Beispielnetzwerk.

Korrektheit

Die Korrektheit des Algorithmus können wir durch die beiden folgenden Formeln spezifizieren:

[1]Die Addition zweier Fragmente bezeichnen wir mit dem Zeichen +. Eine Verwechslung mit der Addition natürlicher Zahlen oder der Multimengenaddition kann nicht entstehen, da immer aus dem Kontext hervorgeht, welche Addition gemeint ist.

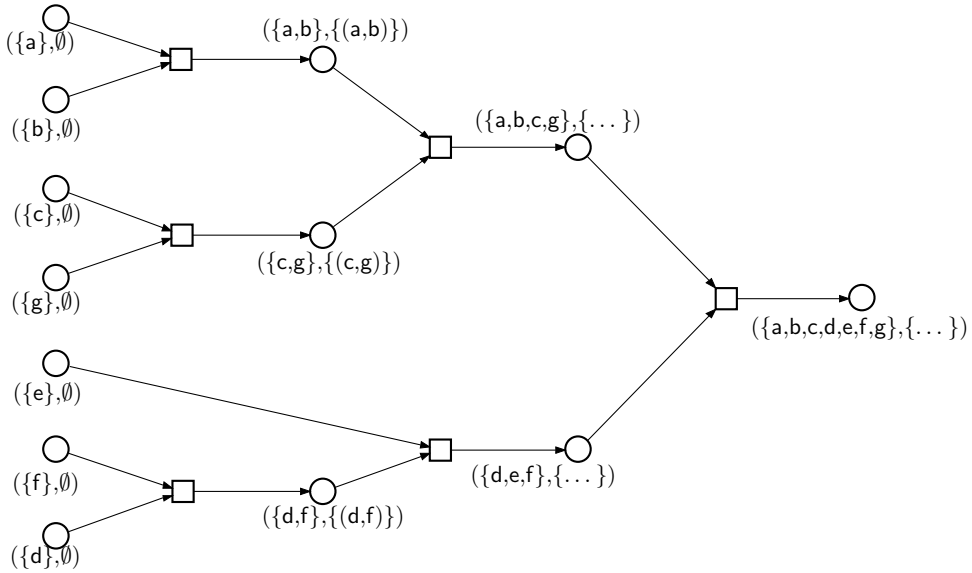


Abb. 7.2: Ein verteilter Ablauf von Σ_1

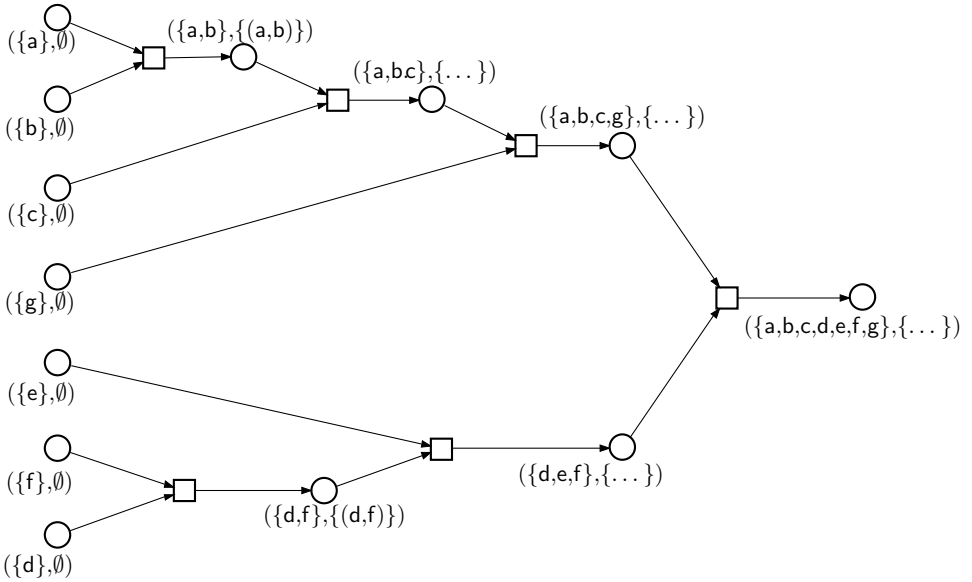


Abb. 7.3: Noch ein verteilter Ablauf von Σ_1

$$\Box\left(|\mathsf{fragments}| = 1 \longrightarrow \mathsf{fragments} = \mathrm{mst}(A, N, w)\right) \tag{S}$$

$$\Diamond\, |\mathsf{fragments}| = 1 \tag{L}$$

Die Formel (S) ist eine Sicherheitseigenschaft (siehe Abschnitt 4.4) und bedeutet: Wenn es nur noch ein Fragment gibt, ist dieses Fragment der gesuchte minimale spannende Baum. Die Formel (L) ist eine Lebendigkeitseigenschaft und sagt aus, daß es irgendwann nur noch ein Fragment gibt.

Wir notieren einige Eigenschaften des Algorithmus, die sich formal am Modell überprüfen lassen:

(i) In jedem erreichbaren Zustand ist jede Marke auf der Stelle fragments ein Fragment. (Die Aussage gilt im Anfangszustand und bleibt wegen Satz 6.4 bei jedem Schalten der Transition erhalten.)

(ii) Jeder Agent gehört immer zu genau einem Fragment. (Die Aussage gilt im Anfangszustand und bleibt bei jedem Schalten der Transition erhalten.)

(iii) Wenn es nur noch eine Marke gibt, ist der Algorithmus terminiert. (t benötigt zwei Marken, um zu schalten.)

(iv) Die Anzahl der Marken auf der Stelle fragments nimmt bei jedem Schalten von t um eins ab.

(v) Wenn es noch mehr als eine Marke gibt, ist t aktiviert. (Wegen (i),(ii) und der Tatsache, daß das Netzwerk zusammenhängend ist.)

Die Sicherheitseigenschaft (S) folgt aus (i), (ii) und (iii). Die Lebendigkeitseigenschaft (L) folgt aus (iv), (v) und der Tatsache, daß die Menge der Knoten des Netzwerks nicht leer ist.

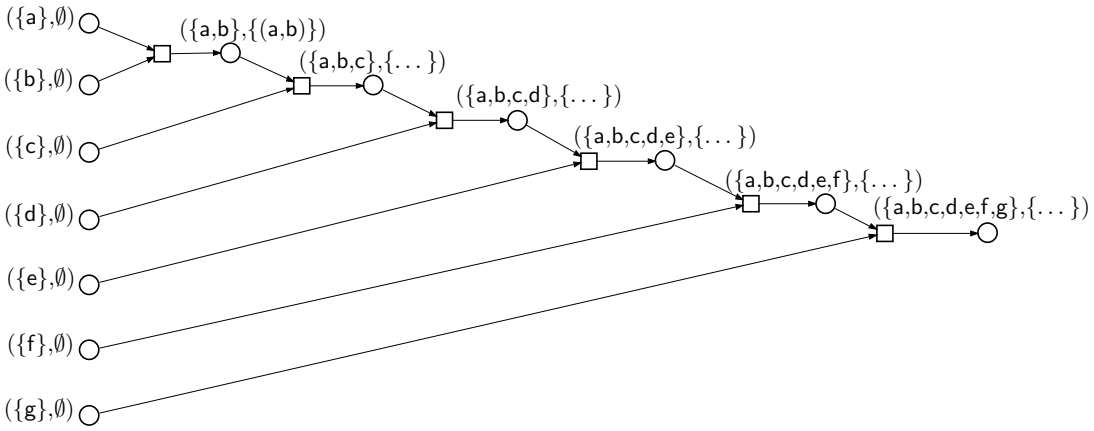


Abb. 7.4: Ein Ablauf der Länge 6.

Ausblick

Wir betrachten noch einmal die verteilten Abläufe von Σ_1. Wir definieren die *Länge eines verteilten Ablaufs* als Anzahl der Ereignisse auf dem längsten Weg in diesem Ablauf. Jeder Ablauf von Σ_1 ist endlich, aber es gibt unterschiedlich lange Abläufe. Der Ablauf in Abb. 7.2 hat die Länge 3. Der Ablauf in Abb. 7.3 hat die Länge 4. Die längsten Abläufe erhält man, indem man sich vorstellt, daß immer nur ein Agent richtig aktiv ist. Das bedeutet, daß immer nur das Fragment dieses Agenten, seine kleinste ausgehende Kante sucht und sich dann mit dem Fragment am anderen Ende der Kante zusammenschließt. Wir betrachten also z. B. den Ablauf von Σ_1, in dem a immer zu einem der beiden Fragmente gehört, die sich zu einem größeren zusammenschließen (siehe Abb. 7.4). Da jeder Baum auf der Menge der Agenten A genau $|A| - 1$ Kanten hat, haben die längsten Abläufe von Σ_1 die Länge $|A| - 1$, in unserem Beispiel also 6.

Unser erster Verfeinerungsschritt besteht darin, solche "langen" (aber auch einige andere) Abläufe auszuschließen, indem wir den Algorithmus etwas mehr synchronisieren.

7.2 Einführung von Leveln

Informelle Beschreibung

Bevor wir Algorithmus 2 erklären, betrachten wir einen Algorithmus, der auf Runden basiert: In jeder Runde sucht jedes Fragment seine kleinste, aus dem Fragment herausführende Kante und verbindet sich über diese Kante mit einem anderen Fragment. Erst wenn sich alle Fragmente über ihre kleinste Kanten mit einem anderen Fragment verbunden haben, wird neu gesucht. Dadurch wird jedes Fragment in jeder Runde zur Aktivität gezwungen und ein langer Ablauf wie in Abb. 7.4 wird ausgeschlossen.

Die Einführung von solchen Runden ist eine starke Synchronisation, die wenig Nebenläufigkeit zuläßt. In Algorithmus 2 führen wir auch eine Art Rundennummer für jedes Fragment ein, die wir den *Level* des Fragments nennen. Wann sich zwei Fragmente zusammenschließen können, hängt nun auch vom Level der Fragmente ab. Wir synchronisieren also die Verbindungen der Fragmente über den Level, jedoch nicht ganz so stark wie gerade beschrieben.

Zu Beginn haben alle Fragmente den Level 0. Zwei Fragmente können sich auf zwei verschiedene Arten zusammenschließen:

(i) Beide Fragmente haben denselben Level l und die gleiche kleinste ausgehende Kante (a, b) bzw. (b, a). Das neu entstehende Fragment hat dann den Level $l + 1$.

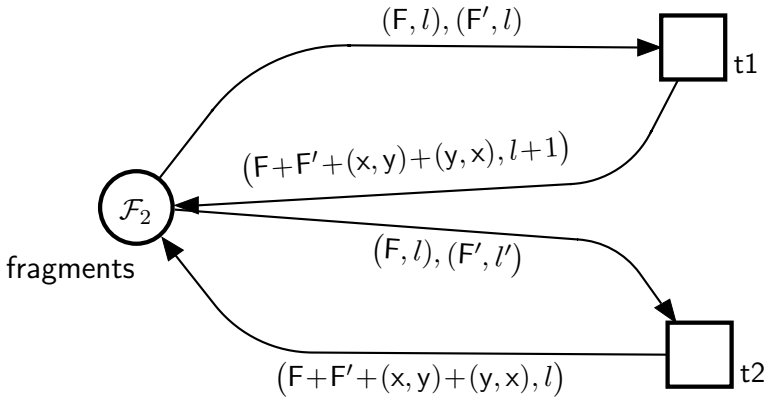
(ii) Die Fragmente haben unterschiedliche Level und die kleinste ausgehende Kante des Fragments mit dem kleineren Level verbindet die beiden Fragmente. Der Level des neuen Fragments ist gleich dem größeren Level der beiden Fragmente.

Den ersten Fall, in dem beide Fragmente gleichberechtigt sind, nennen wir die *Vereinigung* der beiden Fragmente. Im zweiten Fall sprechen wir vom *Beitritt* des Fragments mit dem kleineren Level zum Fragment mit dem größeren Level.

Wir werden später an unserem Beispiel sehen, daß Algorithmus 2 einige Abläufe von Algorithmus 1 ausschließt. Die Bedeutung der Level im GHS-Algorithmus wird aber erst in einem späteren Verfeinerungsschritt (Algorithmus 11) sichtbar. Dort hängt jeder Agent an jede Nachricht seinen Level an. Für den Empfänger der Nachricht ist der Level dann ein Gradmesser für die Aktualität der Nachricht. Wenn der Level in der Nachricht kleiner ist, als sein eigener Level, dann ist die Nachricht möglicherweise veraltet und wird deshalb nicht mehr verwendet.

Das Petrinetzmodell

Algorithmus 2 ist in Abb. 7.5 als Petrinetz dargestellt. Jedes Fragment hat eine neue Variable l, die den Level des Fragments angibt. Der Level ist immer eine natürliche



Sorten		Konstante
l, l' :	$\mathbb{N}$	$\mathcal{F}_2 = A \times \{\emptyset\} \times \{0\}$
fragments :	$\mathbb{F}\text{ragment} \times \mathbb{N}$	

Die anderen Sorten wie in Σ_1.

Aktivierungsbedingungen

t1 : $g_1 \wedge (x, y) = \text{mo}(F)$
t2 : $g_1 \wedge l' < l$

Dabei ist $g_1 \equiv x \in F \wedge y \in F' \wedge (y, x) = \text{mo}(F')$ die Aktivierungsbedingung aus Σ_1.

Abb. 7.5: Σ_2

Zahl. Die Stelle fragments hat deshalb die Sorte $\mathbb{F}\text{ragment} \times \mathbb{N}$. Die Bedingungen für die beiden Arten des Zusammenschlusses sind einerseits durch die Aktivierungsbedingungen und andererseits durch die Kanteninschrift der Kante zu t_1 gegeben, die besagt, daß die Level der beiden Fragmente gleich sein müssen, da die gleiche Variable verwendet wurde.

Betrachten wir noch einmal unser Beispielnetzwerk. Ein verteilter Ablauf von Σ_2 ist in Abb. 7.6 angegeben. Dieser Ablauf entspricht dem Ablauf aus Abb. 7.3. Wenn man in der Beschriftung im Ablauf 7.6 die Level wegläßt, erhält man genau Ablauf 7.3. In beiden Abläufen entstehen die gleichen Fragmente in der gleichen kausalen Halbordnung.

Die Abläufe aus Abb. 7.2 und 7.4 haben im Algorithmus 2 keine Entsprechung. Sie fallen durch die Synchronisation über die Level weg. Im Ablauf in Abb. 7.2 werden die trivialen Fragmente der Agenten c und g verbunden. In Algorithmus 2 ist das nicht möglich, da beide Fragmente den Level 0 haben, aber die Kante mit dem Gewicht 9 zwar die kleinste von g ausgehende, aber nicht die kleinste von c ausgehende Kante ist. Im Ablauf in Abb. 7.2 wird das Fragment aus den Agenten $\{a, b, c\}$ mit dem trivialen Fragment des Agenten d verbunden. In Algorithmus 2 ist das nicht möglich, da das Fragment aus $\{a, b, c\}$ bereits den Level 1 und das Fragment aus d den Level 0 hätte, aber die Kante mit dem Gewicht 7 nicht die

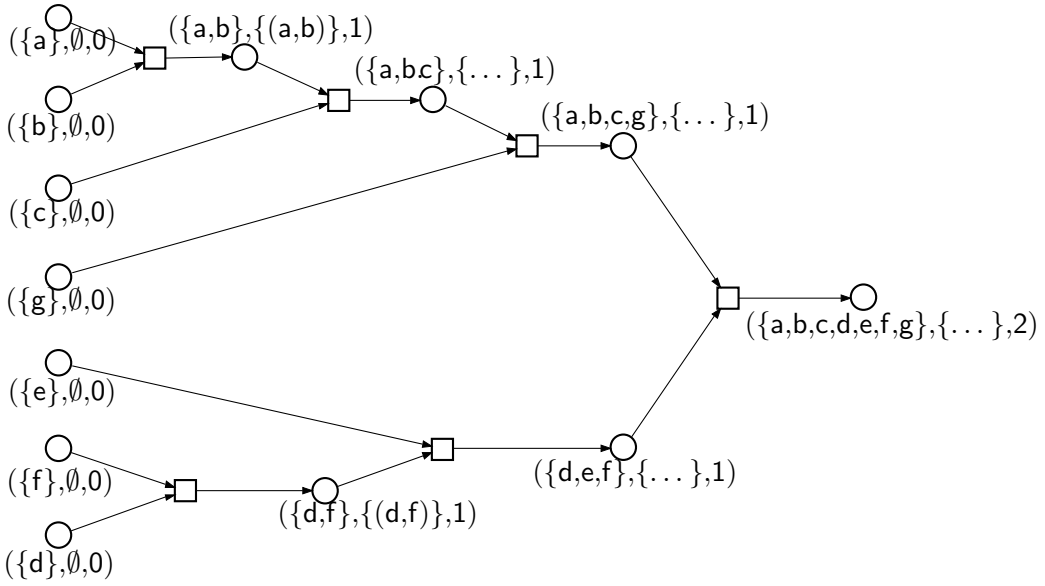


Abb. 7.6: Ein verteilter Ablauf von Σ_2

kleinste von d ausgehende Kante ist.

Für die meisten Kommunikationsnetzwerke hat Σ_2 echt weniger verteilte Abläufe als Σ_1 und es werden unter anderem die "langen" ausgeschlossen. Eine Ausnahme ist z.B. ein Kommunikationsnetzwerk, das einen Stern (vgl. Abschnitt A.2) bildet. In diesem Fall haben Σ_1 und Σ_2 (abgesehen von den Leveln) die gleichen Abläufe.

Korrektheit

Die Korrektheit des Algorithmus können wir analog zu Algorithmus 1 durch die beiden folgenden Formeln spezifizieren[2]:

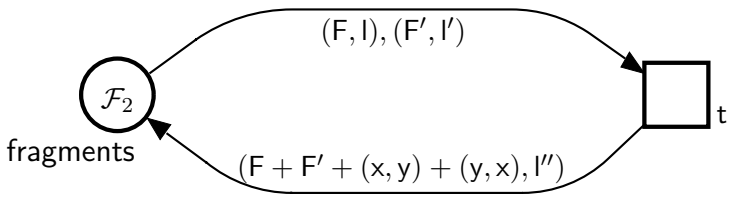
$$\Box\left(|\mathsf{fragments}| = 1 \longrightarrow \mathsf{fragments}_{(1,2)} = \mathrm{mst}(A, N, w)\right) \tag{S}$$

$$\Diamond\, |\mathsf{fragments}| = 1 \tag{L}$$

Wir möchten zeigen, daß jeder Ablauf von Σ_2 ein Ablauf von Σ_1 ist, wenn man die Level ausblendet. Leider sind die Netze verschieden, die Σ_1 und Σ_2 zugrunde liegen. Σ_1 hat nur eine Transition und Σ_2 hat zwei. Σ_2 ist also keine Datenerweiterung von Σ_1.

Wir können das System Σ_2 aber auch anders darstellen. Wir geben in Abb. 7.7 ein System Σ_2' an, in der das zugrunde liegende Netz nur eine Transition hat und die Bedingungen an die Level vollständig in der Aktivierungsbedingung enthalten sind. Wir zeigen dann, daß die Systeme Σ_2 und Σ_2' (bis auf die Beschriftung der Ereignisse) dieselben Abläufe haben und daß das System Σ_2' eine korrekte Datenerweiterung von Σ_1 ist.

[2]Zur Erinnerung: $\mathsf{fragments}_{(1,2)}$ ist einen abkürzende Schreibweise für eine Projektionsfunktion und bedeutet, daß wir nur die ersten beiden Stellen jedes Tripels betrachten. Die Zahlen in der Klammer geben immer an, welche Stellen betrachtet werden.



Sorten und Konstante wie in Σ_2

Aktivierungsbedingung

t : $g_1 \wedge \big((l = l' \wedge (x,y) = \mathrm{mo}(F) \wedge l'' = l + 1) \vee (l' < l \wedge l'' = l)\big)$

Dabei ist $g_1 \equiv x \in F \wedge y \in F' \wedge (y,x) = \mathrm{mo}(F')$ die Aktivierungsbedingung aus Σ_1.

Abb. 7.7: Das Modell Σ_2'

In Σ_2' besteht die Aktivierungsbedingung wieder aus der Aktivierungsbedingung von Σ_1, aber diesmal in Konjunktion mit einer Alternative aus den Aktivierungsbedingungen von $t1$ und $t2$ von Σ_2. In Σ_2 sind einige Bedingungen an die Level bereits in den Kanteninschriften enthalten: Bei $t1$ müssen die Level der beteiligten Fragmente gleich sein. Das wird erreicht, indem bei beiden Fragmenten die gleiche Variable l für den Level verwendet wird. In Σ_2' sind diese Bedingungen Teil der Aktivierungsbedingung:
$\{$ Entweder sind l und l' gleich *und* (x,y) ist auch die kleinste aus F herausführende Kante *und* der Level erhöht sich um eins$\}$ *oder* $\{$ l ist größer als l' *und* der neue Level l'' ist gleich l $\}$.

Wir zeigen nun, daß Σ_2 und Σ_2' dieselben Abläufe haben. Beide Systeme haben die gleiche Anfangsmarkierung. Außerdem können wir jede Aktion (t,β) von Σ_2 kanonisch auf eine Aktion (t',β') von Σ_2' abbilden (und umgekehrt), so daß $(t,\beta)^- = (t',\beta')^-$ und $(t,\beta)^+ = (t',\beta')^+$. Die Systeme Σ_2 und Σ_2' haben damit die gleiche Entfaltung modulo Beschriftung der Transitionen. Nach Satz 4.8 haben damit beide Systeme dieselben Abläufe bis auf die Beschriftung der Ereignisse.

Wir zeigen nun, daß Σ_2' eine korrekte Datenerweiterung von Σ_1 ist. Σ_2' ist eine Datenerweiterung von Σ_1, denn die zugrunde liegenden Netze sind gleich, die Aktivierungsbedingung von Σ_2 impliziert die Aktivierungsbedingung von Σ_2' und die Kanteninschriften sind gleich, wenn man die Level ausblendet. Außerdem können wir jede Markierung M von Σ_2 kanonisch auf eine Markierung von Σ_1 abbilden, indem wir die Level weglassen:

$$f(M)(\mathsf{fragments}) = \mathsf{fragments}_{(1,2)}$$

Σ_2' ist keine konservative Datenerweiterung, denn die neuen Variablen haben Einfluß auf die Aktivierungsbedingungen. Es hängt von den initialen Werten der Level ab, ob das System verklemmt oder nicht. Der Algorithmus funktioniert nur, weil wir initial alle Level gleich Null setzen: Betrachten wir z.B. ein Kommunikationsnetzwerk aus nur zwei Agenten und einer Kante. Wenn wir den beiden trivialen Fragmenten initial unterschiedliche Level zuweisen, kann keine Transition schalten. Das System Σ_2' verklemmt.

Das System Σ_2' ist jedoch eine korrekte Datenerweiterung von Σ_1. Die Anfangsmarkierungen der Systeme stimmen bis auf die Level überein. Nach Satz 5.16 müssen wir nun noch zeigen, daß kein Ablauf von Σ_2' zu früh abbricht und dadurch die Lebendigkeitseigenschaft verletzt. Dazu betrachten wir einen beliebigen erreichbaren Zustand M von Σ_2', in dem es noch mehr als ein Fragment gibt. Die Transition $\mathtt{t}$ aus Σ_1 ist in $f(M)$ aktiviert und wir müssen zeigen, daß $\mathtt{t}$ in M in Σ_2' aktiviert ist.

Dazu betrachten wir alle Fragmente mit minimalem Level in M. Jedes dieser Fragmente hat eine kleinste aus dem Fragment herausführende Kante. Entweder gibt es ein Fragment mit minimalem Level, dessen kleinste aus dem Fragment herausführende Kante in ein Fragment mit höherem Level führt. Dann kann Transition $\mathtt{t}$ in Σ_2' schalten. Oder aber alle kleinsten herausführenden Kanten führen wieder in Fragmente mit minimalem Level. Diese Kanten können keinen Kreis bilden, da die Gewichte von einem zum nächsten Fragment immer kleiner werden. Es gibt also zwei Fragmente mit minimalem Level, die die gleiche kleinste Kante haben und Transition $\mathtt{t}$ in Σ_2' kann auch schalten.

Wir haben gezeigt, daß Σ_2' eine korrekte Datenerweiterung von Σ_1 ist. Da Σ_2 und Σ_2' (bis auf Beschriftung der Ereignisse) die gleichen Abläufe haben und die neue Variable nicht in der Spezifikation der Korrektheit vorkommt, folgt die Korrektheit von Σ_2 aus der Korrektheit von Σ_1.

7.3 Kommunikation zwischen den Fragmenten

Informelle Beschreibung

In Algorithmus 3 beschreiben wir die Kommunikation zwischen den Fragmenten durch Nachrichtenaustausch. Jedes Fragment bestimmt seine kleinste aus dem Fragment herausführende Kante und schickt dann über diese Kante eine Nachricht, um dem Fragment am anderen Ende der Kante anzuzeigen, daß es sich mit ihm verbinden möchte.

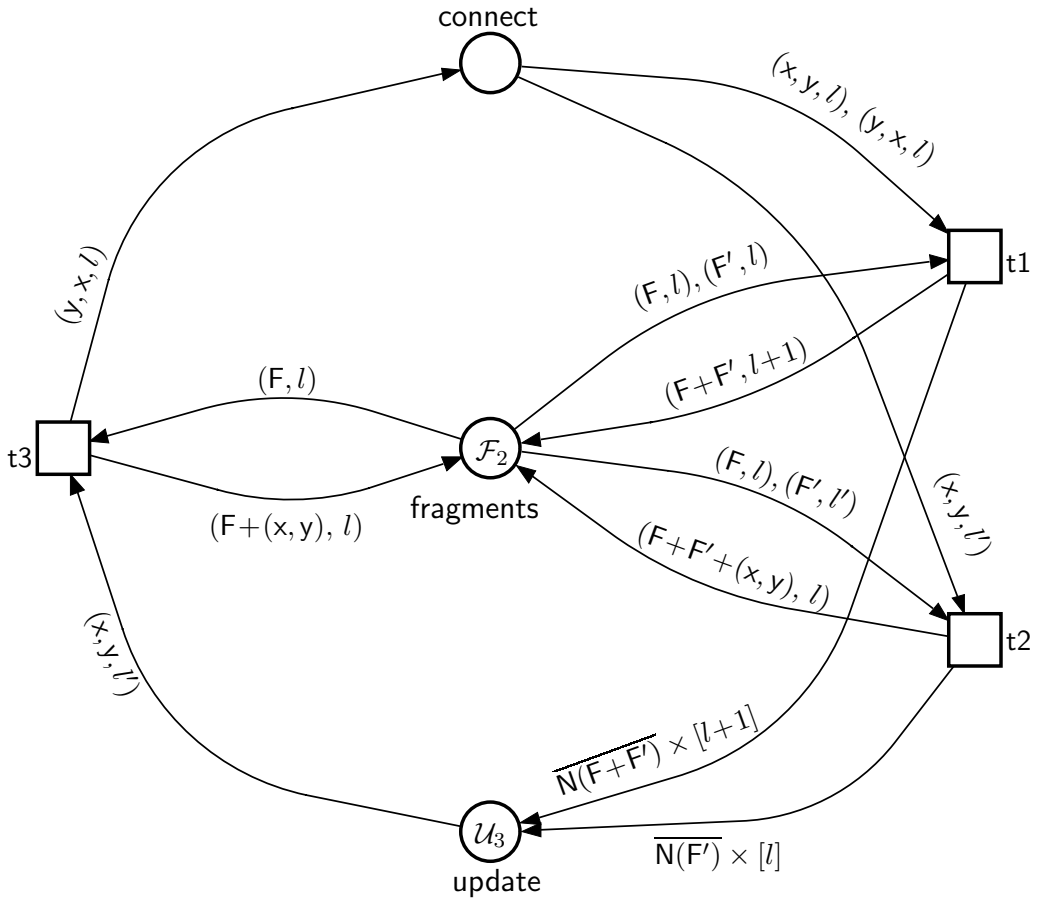
Wenn wir noch einmal Algorithmus 2 betrachten, dann stellen wir fest, daß das Fragment, über dessen kleinste Kante eine Verbindung stattfindet, immer einen höchstens genauso großen Level hat, wie das andere an der Verbindung beteiligte Fragment. Diese Synchronisation wollen wir in Algorithmus 3 beibehalten.

Wenn ein Fragment F seine kleinste ausgehende Kante bestimmt hat und das Fragment F' am anderen Ende der Kante einen mindestens genauso großen Level wie das Fragment selbst hat, dann schickt F eine Nachricht über diese Kante, um dem Fragment am anderen Ende der Kante anzuzeigen, daß es sich mit ihm verbinden möchte. Wenn sich ein neues Fragment bildet, schickt dieses Fragment Nachrichten an alle Nachbarfragmente, um diese über seinen neuen Level zu informieren.

Wie in Algorithmus 2 vereinigen sich zwei Fragmente, wenn sie die gleichen Level haben und sich gegenseitig Nachrichten mit einem Verbindungswunsch zugeschickt haben. Ein Fragment tritt einem Fragment mit höherem Level bei, nachdem es seinen Verbindungswunsch in einer Nachricht an das Fragment mit dem höheren Level angezeigt hat.

Wir gehen in diesem Algorithmus noch nicht darauf ein, *wie* die kleinste Kante bestimmt wird, d.h. wir beschreiben noch nicht die Kommunikation der Agenten innerhalb eines Fragments.

Das Petrinetzmodell



Sorten		Konstanten
connect:	$N \times \mathbb{N}$	$\mathcal{U}_3 = N \times \{0\}$
update:	$N \times \mathbb{N}$	

Die anderen Sorten und $\mathcal{F}_2$ wie in Σ_2.

Aktivierungsbedingungen

t1 : $x \in F \wedge y \in F'$
t2 : $x \in F \wedge y \in F' \wedge l' < l$
t3 : $(x, y) \notin F \wedge (x, y) = \mathrm{mo}(F) \wedge l \leq l'$

Abb. 7.8: Σ_3

Wir erklären Algorithmus 3 nun formal anhand des Petrinetzmodells in Abb. 7.8.

Eine Nachricht hat in unserem Modell die Form (y, x, l), wobei y einen Agenten des Empfängerfragments, x einen Agenten des Absenderfragments und l den Level des Absenderfragments bezeichnet. Der Kanal, über den die Nachricht (y, x, l) versendet wird, ist die Kante (x, y). Wir unterscheiden connect- und update-Nachrichten. Eine connect-Nachricht zeigt einen Verbindungswunsch an und eine update-Nachricht informiert das Empfängerfragment über den Level des Absenderfragments.

Da zu Beginn alle Fragmente den Level 0 haben, geben wir in der Anfangsmarkierung eine update-Nachricht mit dem Level 0 für jede Kante des Netzwerks vor[3]. Zu Beginn ist für jedes triviale Fragment Transition t3 aktiviert.

Transition t3: Ein Fragment schickt über seine kleinste aus dem Fragment herausführende Kante eine connect-Nachricht, um dem Fragment am anderen Ende der Kante anzuzeigen, daß es sich mit ihm verbinden möchte. Eine connect-Nachricht darf jedoch erst dann abgeschickt werden, wenn der Level des Empfängerfragments mindestens so groß wie der eigene Level ist. Dies ist der Fall, wenn eine update-Nachricht vom Empfängerfragment mit ausreichend großem Level da ist. Ein Fragment merkt sich, daß es eine connect-Nachricht über die Kante (x, y) verschickt hat, indem es (x, y) in seine Kantenmenge aufnimmt. In der Aktivierungsbedingung der Transition t3 sind die bereits erwähnten Bedingungen zusammengefaßt:

- (x, y) gehört noch nicht zur Kantenmenge des Fragments.
- (x, y) ist die kleinste aus dem Fragment herausführende Kante.
- Es liegt eine Nachricht des Empfängerfragments vor, deren Level mindestens so groß ist, wie der Level des Absenderfragments.

Wie in Σ_2 modellieren die Transitionen t1 und t2 die beiden Arten der Verbindung zweier Fragmente.

Transition t1: Zwei Fragmente vereinigen sich, wenn beide Fragmente denselben Level haben und sich gegenseitig connect-Nachrichten zugeschickt haben. Das neu entstehende Fragment erhöht seinen Level um eins und sendet update-Nachrichten an alle benachbarten Fragmente[4].

Transition t2: Ein Fragment, welches eine connect-Nachricht an ein Fragment mit einem höheren Level als seinem eigenen geschickt hat, tritt diesem Fragment bei. Dabei behält das neue Fragment den höheren Level beider Fragmente und es werden update-Nachrichten mit dem höheren Level an alle benachbarten Fragmente des Fragments mit dem kleineren Level verschickt.

Die Aktivierungsbedingungen der Transitionen t1 und t2 sind hier schwächer als in Σ_2, da die Bestimmung der kleinsten Kante bereits vor der Verbindung zweier

[3] Dies ist ein Trick zur gefälligeren Modellierung. Die update-Nachrichten sind Hilfsvariablen, die in einem späteren Verfeinerungsschritt wieder verschwinden. Wir kümmern uns deshalb nicht darum, wie die Nachrichten in der Anfangsmarkierung entstehen.

[4] Genaugenommen werden sogar mehr update-Nachrichten versendet: Eine Nachricht an jeden Nachbarn jedes Agenten des Fragments, unabhängig davon, ob der Nachbar bereits zum Fragment gehört oder nicht. Die zuviel verschickten Nachrichten bleiben einfach auf der Stelle update liegen. Wir haben uns für diese Modellierung auch im Hinblick auf zukünftige Verfeinerungsschritte entschieden: Dann versendet jeder Agent einzeln update-Nachrichten, ohne von jedem seiner Nachbarn zu wissen ob dieser zu seinem Fragment gehört oder nicht.

Fragmente stattfindet. Außerdem nimmt beim Schalten der Transition t3 ein Fragment seine kleinste aus dem Fragment herausführende Kante (x, y) bereits in seine Kantenmenge auf, um sich zu merken, daß über diese Kante eine connect-Nachricht verschickt wurde. Im mathematischen Sinn ist es dann kein Fragment mehr, denn ein Fragment ist ein Baum, d.h. eine Menge von Agenten und Kanten zwischen diesen Agenten. Wir haben aber nun eine Kante (x, y) in der Kantenmenge, ohne daß der Agent y zur Agentenmenge des Fragments gehört. Im formalen Beweis müssen wir dies natürlich berücksichtigen, im Text werden wir in diesem Fall jedoch weiterhin den Begriff Fragment benutzen.

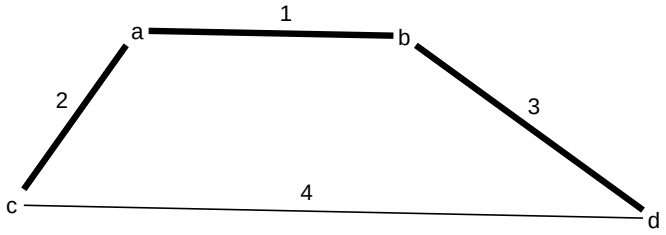


Abb. 7.9: Kommunikationsnetzwerk mit minimalem spannenden Baum

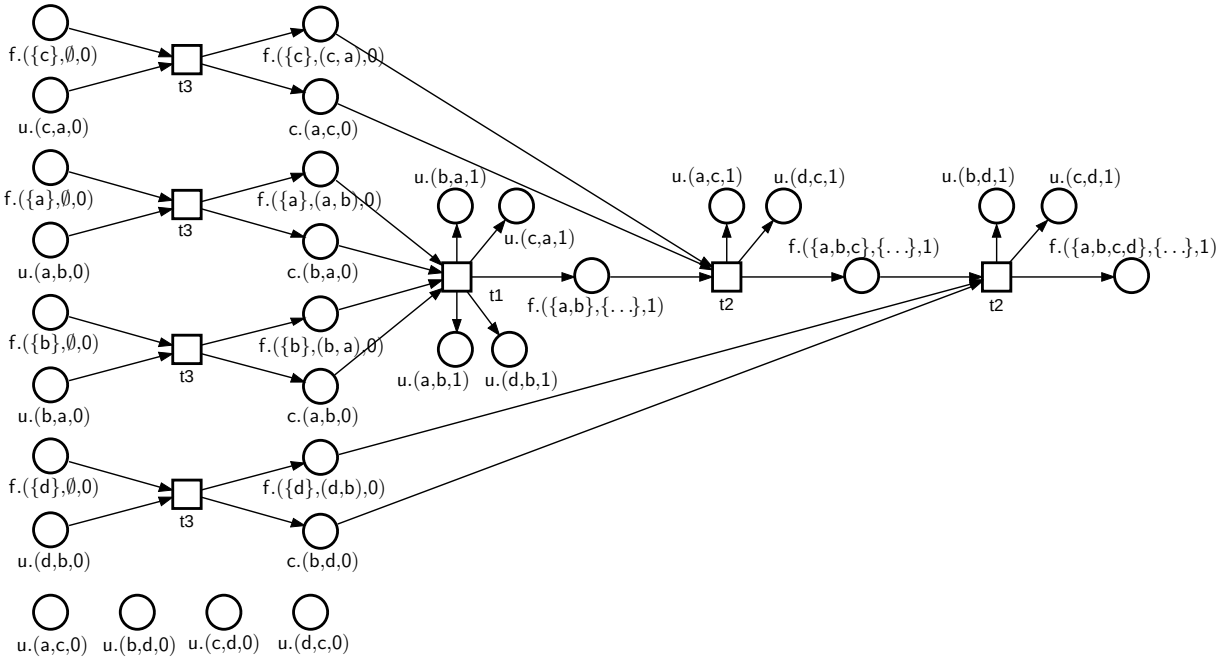


Abb. 7.10: Ein verteilter Ablauf von Σ_3

Wir betrachten nun das Kommunikationsnetzwerk in Abb. 7.9. Der minimale spannende Baum des Netzwerks ist fett gedruckt. In Abb. 7.10 und Abb. 7.11 sind zwei verteilte Abläufe von Σ_3 für dieses Netzwerk dargestellt. Aus Platzgründen haben wir die Beschriftung der Bedingungen mit dem Anfangsbuchstaben der Stellen abgekürzt. In beiden Abläufen schickt zuerst jedes triviale Fragment unabhängig von den anderen Fragmenten eine connect-Nachricht über seine kleinste herausführende Kante. Dann vereinigen sich die trivialen Fragmente von a und b zu einen Fragment mit dem Level 1. Im ersten Ablauf tritt zuerst das triviale Fragment von c diesem

Fragment bei, danach tritt das triviale Fragment von d bei. Im zweiten Ablauf ist es umgekehrt.

Die isolierten Bedingungen sind update-Nachrichten, die in der Anfangsmarkierung bereitgestellt, aber nicht verbraucht wurden. Auch die update-Nachrichten, die während des Ablaufs entstehen, werden nicht verbraucht. Ein anderer verteilter Ablauf entsteht, wenn das triviale Fragment $(\{c\}, \emptyset)$ zum Schalten der Transition t3 die Nachricht $\mathsf{u.(c, a, 1)}$ verbraucht und dafür die Nachricht $\mathsf{u.(c, a, 0)}$ liegenbleibt.

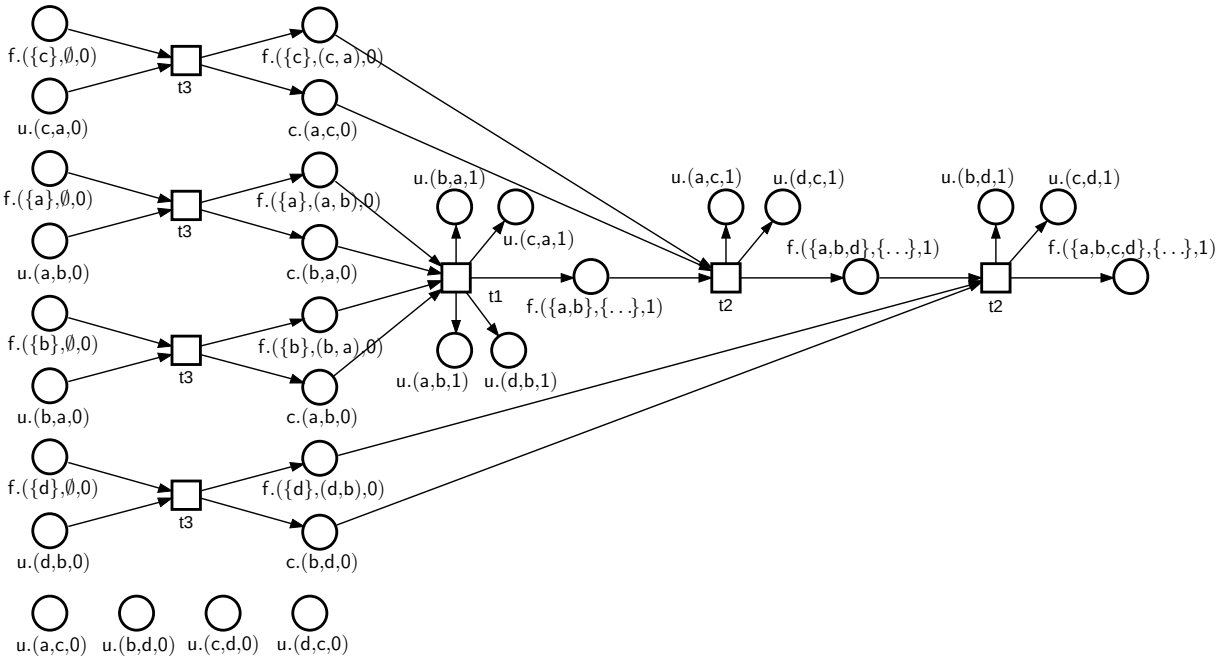


Abb. 7.11: Ein anderer verteilter Ablauf von Σ_3

Korrektheit

Die Korrektheit von Algorithmus 3 spezifizieren wir genau so wie die von Algorithmus 2. Wir wählen nun Beobachter von Σ_2 und Σ_3, so daß Σ_3 eine Beobachterverfeinerung von Σ_2 ist. Dabei sollen die Beobachter soviel sehen, daß die Korrektheit von Σ_3 aus der Korrektheit von Σ_2 folgt.

Wir definieren den Beobachter von Σ_2 als $\mathrm{obs}_2 = \mathsf{fragments}$. Der Beobachter sieht also die gesamte Markierung. Der Beobachter von Σ_3 sieht auch die Stelle fragments, aber er ignoriert die Kanten des Netzwerks, über die gerade eine connect-Nachricht unterwegs ist. Diese Kanten sieht er wie in Algorithmus 2 erst dann, wenn die Verbindung von zwei Fragmenten wirklich stattgefunden hat. Wenn Transition t3 schaltet, ändert sich für diesen Beobachter nichts, wenn dagegen Transition t1 oder t2 schaltet, sieht der Beobachter dieselbe Veränderung wie der Beobachter von Σ_2, wenn t1 oder t2 im gleichen Modus schaltet. Als Beobachter von Σ_3 definieren wir also

$$\mathrm{obs}_3 = \big\{(V, M \setminus \mathsf{connect}_{(2,1)}, l) \mid \text{ für alle } (V, M, l) \in \mathsf{fragments}\big\},$$

wobei V eine Menge von Agenten, M eine Menge von Kanten und $\setminus$ die normale Mengensubtraktion bezeichnen.

Wir können zeigen, daß Σ_2 das System Σ_3 bezüglich dieser Beobachter mit den Kriterien aus Satz 5.22 simuliert. Dazu benötigen wir einige Invarianten von Σ_3.

$$\Box \mathsf{fragments}_{(1)} = A \tag{7.1}$$

$$\Box \mathsf{fragments}(F, l) \wedge (x, y) = \mathrm{mo}(F) \rightarrow (\mathsf{connect}(y, x, l) \vee (x, y) \notin F) \tag{7.2}$$

$$\Box \mathsf{fragments}(F, l) \wedge \mathsf{connect}(y, x, l') \wedge x \in F \rightarrow (l = l' \wedge y \notin F \wedge (x, y) = \mathrm{mo}(F)) \tag{7.3}$$

Diese Invarianten kann man überprüfen, indem man feststellt, daß sie in der Anfangsmarkierung gelten und wenn sie in einer erreichbaren Markierung vor dem Schalten einer Transition gelten, dann gelten sie auch noch danach.

Wir wissen bereits, daß Σ_2 nur endliche Abläufe hat. Die Transition t_3 kann in Σ_3 nur endlich oft schalten, da bei jedem Schalten eine Baumkante zu einem Fragment hinzugefügt wird, die vorher noch keine Baumkante war. Potentiell gibt es nur endlich viele Baumkanten und von keiner anderen Transition werden Baumkanten wieder abgezogen. Wenn wir also mit den anderen Kriterien zeigen, daß $t1$ und $t2$ in Σ_3 von Σ_2 simuliert werden können, dann wissen wir, daß auch Σ_3 nur endliche Abläufe hat. Wir zeigen nun nacheinander die Gültigkeit aller Kriterien von Satz 5.22.

(i) Die beobachtbaren Anfangsmarkierungen beider Systeme stimmen überein.

(ii) (a) Wenn $t3$ schaltet, ändert sich die beobachtbare Markierung von Σ_3 nicht.

(b) Wenn in einer erreichbaren Markierung M_3 von Σ_3 die Transition $t1$ aktiviert ist und für die Markierung M_2 von Σ_2 gilt $\mathrm{obs}_2(M_2) = \mathrm{obs}_3(M_3)$, dann ist nach Invariante (7.3) auch $t1$ in Σ_2 im gleichen Modus aktiviert und nach dem Schalten sind die beobachtbaren Zustände wieder gleich. Analog gilt diese Argumentation für $t2$.

(iii) Sei M_3 eine tote erreichbare Markierung in Σ_3. Sei außerdem M_2 eine Markierung von Σ_2 für die $\mathrm{obs}_2(M_2) = \mathrm{obs}_3(M_3)$ gilt. Angenommen, in M_2 ist $t1$ aktiviert. Dann ist wegen Invariante (7.2) auch $t1$ in M_3 aktiviert, aber dies ist ein Widerspruch dazu, daß M_3 tot ist. Analog gilt diese Argumentation für $t2$.

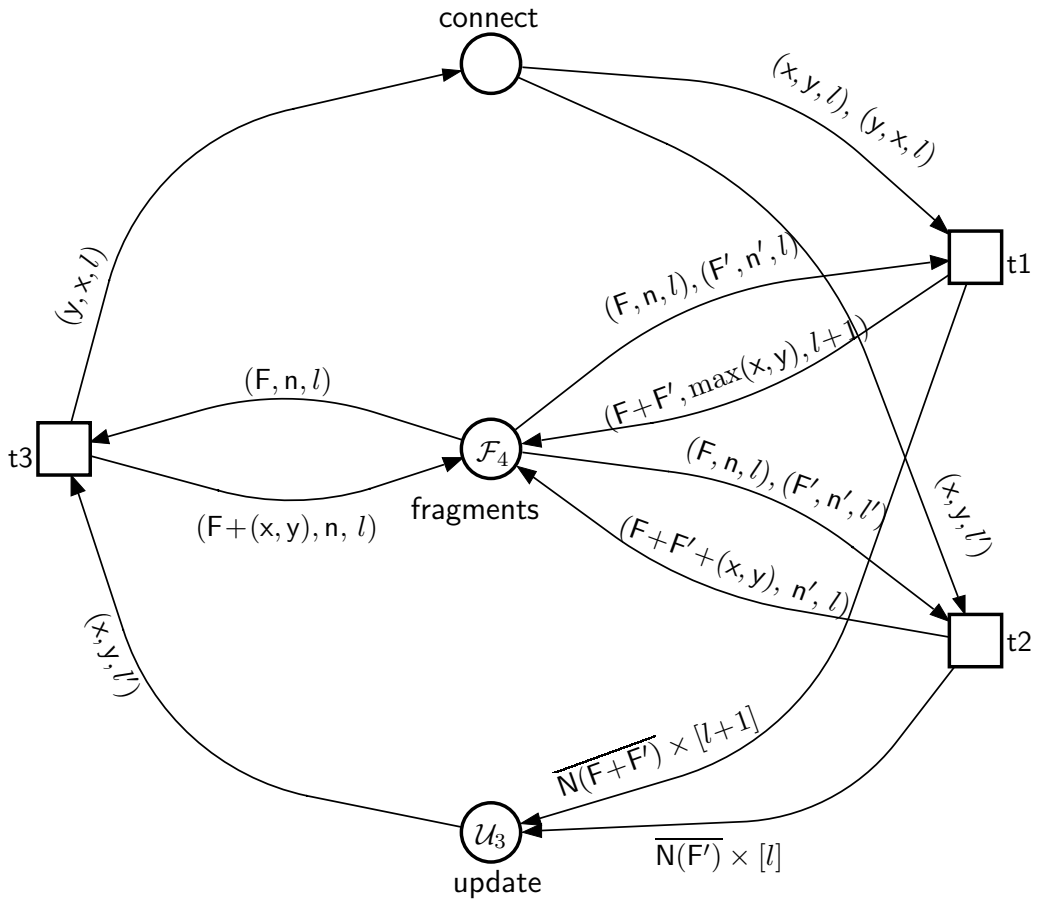
Aus der Beobachterverfeinerung können wir schließen, daß es auch in Algorithmus 3 irgendwann nur noch ein Fragment gibt und daß dieses Fragment der minimale spannende Baum ist, wenn man die connect-Nachrichten abzieht, die noch unterwegs sind. Wir müssen also noch zeigen, daß keine connect-Nachricht mehr unterwegs ist, wenn es nur noch ein Fragment gibt. Dies folgt aus den Invarianten (7.1) und (7.3).

7.4 Fragmentnamen

Informelle Beschreibung

Unser nächstes Ziel ist es, die Daten lokal den Agenten zuzuordnen, statt sie als ganzes Fragment zu speichern. Dazu muß jeder Agent auch wissen, zu welchem Fragment er gehört. Deshalb führen wir als vorbereitenden Schritt Namen für Fragmente ein.

Der Algorithmus 4 unterscheidet sich vom Algorithmus 3 nur dadurch, daß jedes Fragment eine zusätzliche Variable n hat, die den Namen des Fragments angibt. Der Name eines Fragments ist immer der Name eines Agenten des Fragments. Wir nennen den Agenten n den *Namensgeber* des Fragments.



Sorten		Konstanten
n, n' :	A	$\mathcal{F}_4 = \{(\{a\}, \emptyset, a, 0) \mid a \in A\}$
fragments :	$\mathbb{F}\text{ragment} \times A \times \mathbb{N}$	

Die anderen Sorten, die Konstante und die Aktivierungsbedingungen wie in Σ_3.

Abb. 7.12: Σ_4

Die Fragmentnamen werden nun folgendermaßen berechnet: Das triviale Fragment eines Agenten trägt dessen Namen. Bei einer Vereinigung zweier Fragmente wird der Name des neuen Fragments gerade der Name des größeren[5] der beiden Agenten an der Vereinigungskante. Bei einem Beitritt behält das neue Fragment den Namen des

[5]Hierzu benutzen wir eine zusätzliche Annahme an das Netzwerk: Wir nehmen an, daß es eine irreflexive, partielle Ordnung $\prec \subseteq A \times A$ gibt, durch die Nachbarn miteinander verglichen werden können, d.h. für $(a, b) \in N$ gilt $a \prec b$ oder $b \prec a$. Wir definieren das Maximum zweier Agenten als $\max(a, b) = a$ falls $b \prec a$.

Fragments mit dem höheren Level.

Das Petrinetzmodell

Ein Fragment hat nun einen Namen und einen Level. Eine Marke auf der Stelle fragments hat die Form (V, M, n, l). Dabei ist

- V die Menge der Agenten des Fragments,
- M die Menge der Kanten die schon als Kanten des minimalen spannenden Baumes ermittelt wurden,
- n der Name des Fragments,
- l der Level des Fragments.

Die Bedeutungen der Kanteninschriften am Netz Σ_4 sind dieselben wie in Σ_3, bis auf die neuen Variablen n und n' für Agenten. Auch die Aktivierungsbedingungen sind identisch mit denen aus Σ_3.

Korrektheit

Das System Σ_4 ist eine Datenerweiterung des Systems Σ_3, denn die zugrundeliegenden Netze sind gleich, nur die Sorte der Stelle fragments wurde um eine Komponente erweitert. Nach Satz 5.18 ist es sogar eine konservative Datenerweiterung, da die Aktivierungsfunktionen beider Systeme gleich sind und Variablen für die neue Komponente an allen eingehenden Kanten einer Transition unterschiedlich sind.

Da bei einer konservativen Datenerweiterung das Ausgangsnetz und das erweiterte Netz dieselben Abläufe haben (bis auf Projektion der neuen Variablen), bleiben alle temporalen Eigenschaften erhalten, wenn man die Projektionen anpaßt. Deshalb gelten auch in Σ_4 die Sicherheits- und Lebendigkeitseigenschaft des Algorithmus 4:

$$\Box\left(|\mathsf{fragments}| = 1 \longrightarrow \mathsf{fragments}_{(1,2)} = \mathrm{mst}(A, N, w)\right) \quad \text{(S)}$$

$$\Diamond\, |\mathsf{fragments}| = 1 \quad \text{(L)}$$

Eine einfache Invariante über die Fragmentnamen, die wir in weiteren Verfeinerungsschritten benutzen, ist, daß es in keinem erreichbaren Zustand zwei Fragmente mit dem gleichen Namen gibt. Die Namen sind also eindeutige Identifikatoren der Fragmente. Wir können diese Invariante ausdrücken durch:

$$\Box\left(\mathsf{fragments}(F, n, l) \wedge \mathsf{fragments}(F', n, l') \longrightarrow F = F' \wedge l = l'\right)$$

Diese Invariante folgt unmittelbar aus der Invariante (7.1), die aufgrund der Datenerweiterung erhalten bleibt, und der Invariante

$$\Box\, \mathsf{fragments}(F, n, l) \rightarrow n \in F_{(1)}.$$

7.5 Von Fragmenten zu Agenten: verteilte Daten

Informelle Beschreibung

Bisher waren die Daten im Algorithmus immer Fragmenten zugeordnet. Ein Fragment war eine handelnde Einheit, bestehend aus Agenten und Kanten, die bereits als zum minimalen spannenden Baum gehörend berechnet wurde. Jedem Fragment war ein Name und ein Level zugeordnet.

In diesem Verfeinerungsschritt verteilen wir die Daten auf die einzelnen Agenten. Jeder Agent speichert lokal, welche seiner adjazenten Kanten schon als Kanten des minimalen spannenden Baumes berechnet wurden. Außerdem speichert jeder Agent lokal den Namen und den Level seines Fragments. Ein Fragment ergibt sich implizit als Zusammenfassung aller Agenten, die den gleichen Fragmentnamen gespeichert haben.

Algorithmus 5 arbeitet ähnlich wie Algorithmus 4. Wir fordern bei jeder Aktion, daß alle Agenten eines Fragments die Aktion gemeinsam und gleichzeitig ausführen. Ein Agent ist also eine handelnde Einheit, aber alle seine Aktionen sind mit anderen Agenten synchronisiert. Die Daten in Algorithmus 5 sind zwar verteilt, aber der Algorithmus ist nicht nachrichtenbasiert.

Die einzige algorithmische Veränderung gegenüber Algorithmus 4 ist der Beitritt. An einem Beitritt wirkt von dem Fragment mit dem größeren Level nur noch ein Agent mit. Dieser Agent ist natürlich der Agent an der Beitrittskante, für den ein neuer Baumnachbar hinzukommt. Für alle anderen Agenten des Fragments mit dem größeren Level ändert sich nichts, deshalb brauchen sie auch beim Beitritt nicht mitzuwirken. Bei diesem Algorithmus kann es also passieren, daß zwei Beitritte zu einem Fragment nebenläufig zueinander stattfinden.

Das Petrinetzmodell

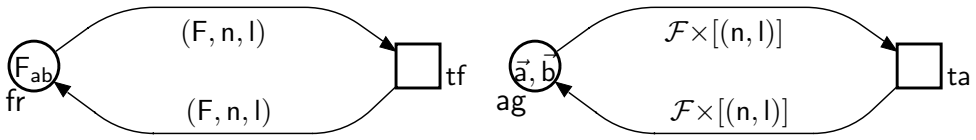
Ein Fragment wurde bisher immer durch eine einzige Marke im Modell repräsentiert. Im Petrinetzmodell von Algorithmus 5 wird ein Fragment durch viele Marken, nämlich für jeden Agenten eine Marke, repräsentiert.

Im Modell von Algorithmus 5 benutzen wir statt der Sorte $\mathbb{F}\text{ragment}$ die Sorte $\mathbb{A}\text{gent} = A \times 2^N$ für Agenten. Im Modell gehört zu einem Agenten x eine Menge von Kanten, die sich im Laufe des Algorithmus verändert. Diese Menge enthält immer die von x ausgehenden Kanten, die schon als Kanten des minimalen spannenden Baumes ermittelt wurden. Am Anfang ist diese Menge für jeden Agenten leer. Eine Marke auf der Stelle agents hat die Form (x, M, n, l). Dabei ist

- x der Name des Agenten,
- M die Menge der von x ausgehenden Kanten die schon als Kanten des minimalen spannenden Baumes ermittelt wurden,
- n der Name des Fragments von x,
- l der Level des Fragments von x.

Zwei Agenten gehören zum selben Fragment, wenn sie den gleichen Namen und den gleichen Level haben.

In Algorithmus 5 führen immer alle Agenten eines Fragments gemeinsam eine Aktion durch. Wir erzwingen dies durch neue Aktivierungsbedingungen, wie im folgenden Beispiel.



Sorten	
F :	$\mathbb{F}$ragment
n :	A
l :	$\mathbb{N}$
fr :	$\mathbb{F}$ragment $\times A \times \mathbb{N}$

Konstanten

$F_{ab} = \big(\{a, b\}, \{(a, b), (b, a)\}, a, 3\big)$

Sorten	
$\mathcal{F}$:	$2^{\mathbb{A}\text{gent}}$
n :	A
l :	$\mathbb{N}$
ag :	$\mathbb{A}$gent $\times A \times \mathbb{N}$

Konstanten

$\vec{a} = (a, \{(a, b)\}, a, 3)$
$\vec{b} = (b, \{(b, a)\}, a, 3)$

Aktivierungsbedingung

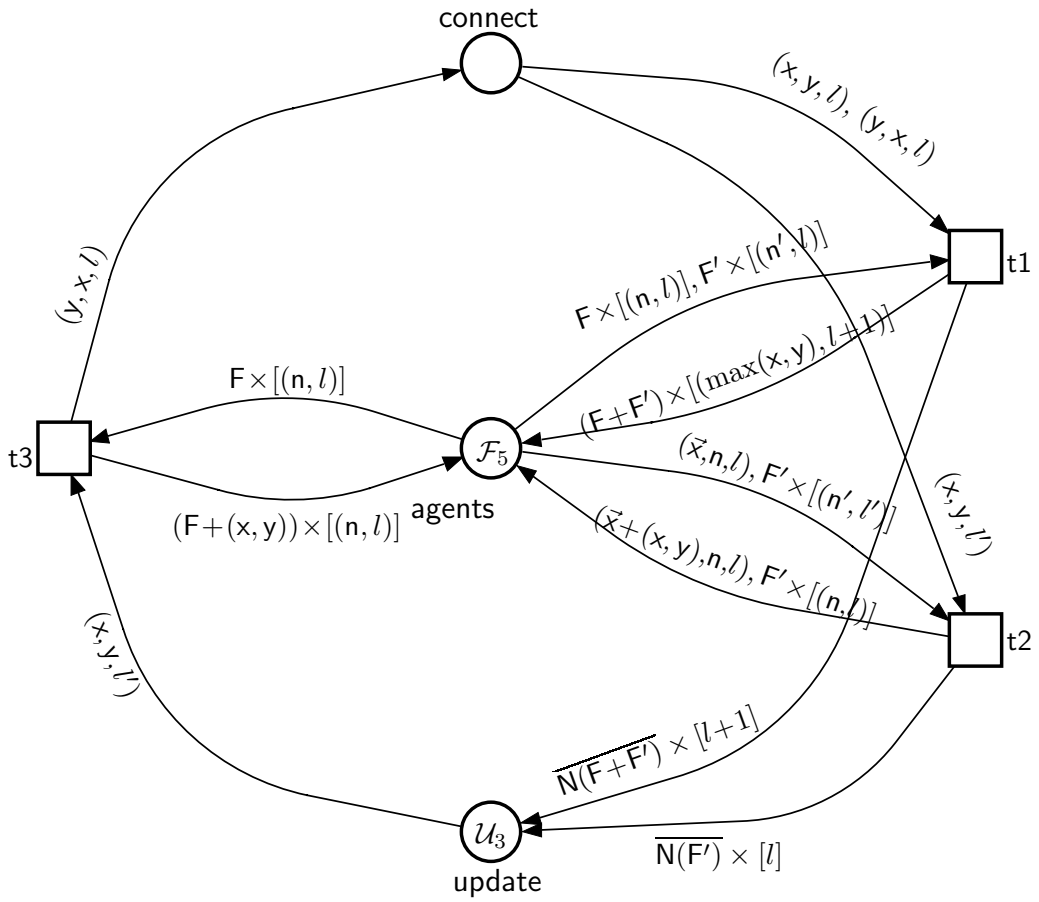
ta : $\forall z, z' : (z, z') \in \mathcal{F}_{(2)} \rightarrow z' \in \mathcal{F}_{(1)}$

Abb. 7.13: Fragmente vs. synchrone Agenten

Wir betrachten ein Fragment, das aus den Agenten a und b und der Kante $(a, b), (b, a)$ besteht. Das Fragment hat den Namen a und den Level 3. In Abb. 7.13 haben wir links eine Transition für das Fragment wie bisher dargestellt, rechts dieselbe Transition mit synchronen Agenten. Rechts wird das Fragment durch die beiden Marken $\vec{a}$ und $\vec{b}$ dargestellt, für jeden Agenten eine Marke. Wir wollen, daß die beiden Agenten des Fragments immer gemeinsam schalten. Dies erreichen wir mit der Aktivierungsbedingung, die aussagt: immer wenn ein Agent z am Schalten beteiligt ist und z' Baumnachbar von z ist, dann ist auch z' am Schalten beteiligt.

Im Petrinetzmodell von Algorithmus 5 verwenden wir nicht $\mathcal{F}, \mathcal{F}'$ als Variablen für Fragmente, sondern weiterhin die Variablen F und F', denen wir eine neue Sorte zuordnen. F ist von nun an eine Variable für eine (Multi-)Menge[6] von geordneten Paaren (x, M), wobei wieder x der Name eines Agenten ist und M die Menge der von x ausgehenden Kanten, von denen bereits ermittelt wurde, daß sie zum minimalen spannenden Baum gehören. Ähnlich wie bisher schreiben wir abkürzend $x \in F$ für einen Agenten x, wenn es eine Menge von Kanten M gibt, so daß $(x, M) \in F$ und wir schreiben $(x, y) \in F$ für eine Kante (x, y), wenn es eine Menge M gibt, so daß $(x, y) \in M$ und $(x, M) \in F$ gilt.

[6] Wir können in diesem, wie auch in allen folgenden Modellen mit einer einfachen Stelleninvariante zeigen, daß es immer für jeden Agenten des Netzwerks genau eine Marke gibt. Das gleiche gilt für Nachrichten: es gibt keine Nachrichtenstelle, auf der gleichzeitig zwei identische Nachrichten liegen. Wir werden deshalb im weiteren über Mengen reden.



Sorten		Konstante
F,F' :	$2^{\mathbb{A}\text{gent}}$	$\mathcal{F}_5 = \{(a,\emptyset,a,0) \mid a \in A\}$
$\vec{x}$:	$\mathbb{A}\text{gent}$	
agents :	$\mathbb{A}\text{gent} \times A \times \mathbb{N}$	

Alle anderen Sorten und $\mathcal{U}_3$ wie in Σ_4.

Aktivierungsbedingungen

t1 : $x \in F \wedge y \in F' \wedge \forall z, z' : (z,z') \in F+F' \to z' \in F+F'$
t2 : $l' < l \wedge y \in F' \wedge \forall z, z' : (z,z') \in F' \to z' \in F' \vee z' = x$
t3 : $(x,y) \notin F \wedge (x,y) = \mathrm{mo}(F) \wedge l \leq l' \wedge \forall z, z' : (z,z') \in F \to z' \in F$

Abb. 7.14: Σ_5

Auch für die neue Sorte von F definieren wir die kleinste aus F herausführende Kante als Kante mit dem kleinsten Gewicht, die einen Agenten aus F mit einem Agenten außerhalb von F verbindet: $\mathrm{mo}(F) = (x,y)$, wenn $x \in F_{(1)}$ und $w(x,y) = \min\{w(x,y) \mid (x,y) \in N(F_{(1)}) \wedge y \notin F_{(1)}\}$.

In den Kanteninschriften bedeutet $F + F'$ normale Multimengenaddition. Die Addition einer Kante zu einem Fragment $F + (x,y)$ ist nur definiert, wenn der Agent

x zum Fragment F gehört. Bei der Addition wird der Kantenmenge von x eine Kante hinzugefügt. Wenn also $F = [\dots, (x, M), \dots]$, dann ist $F + (x, y) = [\dots, (x, M + [(x, y)]), \dots]$. Für $\vec{x} = (x, M)$ ist $\vec{x} + (x, y) = (x, M \cup (x, y))$.

Die Aktivierungsbedingungen von Σ_5 bestehen aus den Aktivierungsbedingungen aus Σ_4 in Konjunktion mit der Bedingung für ta in Abb. 7.13. Eine kleine Änderung an der Aktivierungsbedingung gibt es für t2: Da das beitretende Fragment sich bereits seine kleinste herausführende Kante beim Abschicken der connect-Nachricht gemerkt hat, gehört die Kante (y, x) bereits zum Fragment, der Agent x jedoch noch nicht. x ist deshalb in der Aktivierungsbedingung ausgenommen.

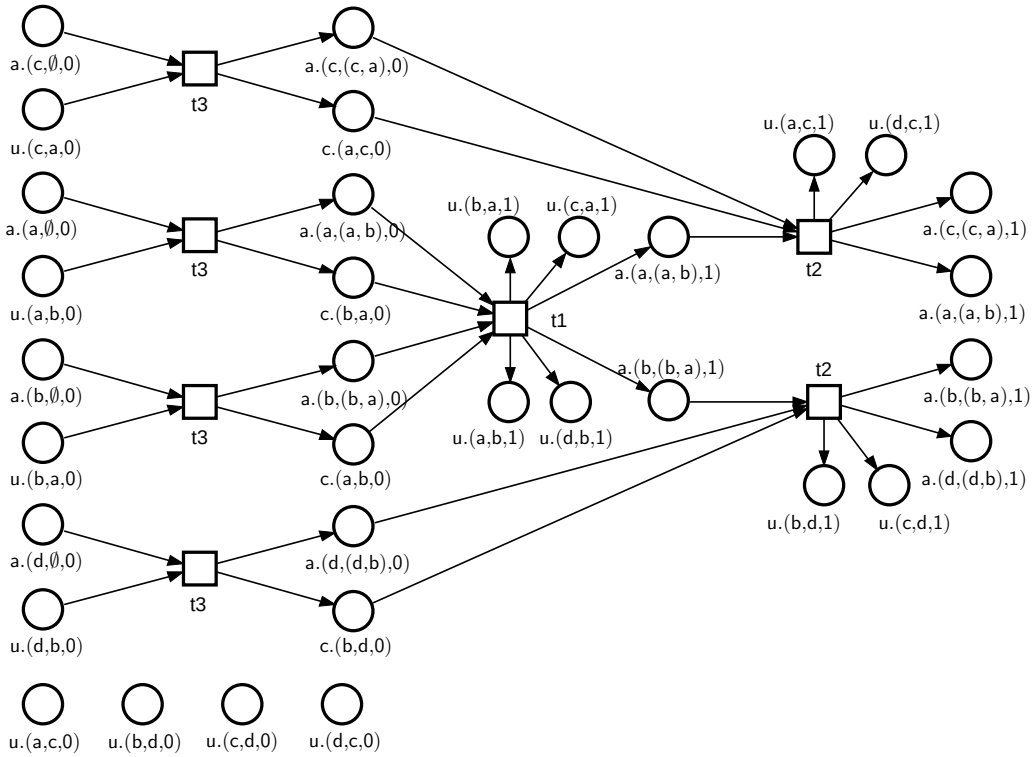


Abb. 7.15: Ein verteilter Ablauf von Σ_5

Abb. 7.15 zeigt einen verteilten Ablauf von Σ_5 für das Netzwerk aus Abb. 7.9. Wie in den Abläufen in Abb. 7.10 und 7.11 vereinigen sich hier die trivialen Fragmente von a und b zu einem größeren Fragment vom Level 1, das dann durch die beiden Marken a.(a,(a, b),1) und a.(b,(b, a),1) repräsentiert wird. Danach treten die trivialen Fragmente von c und d nebenläufig zueinander bei. Im Algorithmus 4 waren diese Beitritte sequentialisiert.

Korrektheit

Formal führen wir in diesem Verfeinerungsschritt eine Beobachterverfeinerung durch. Wir definieren den Beobachter von Σ_4 als

$$\mathrm{obs}_4(M) = (M(\mathsf{fragments}), M(\mathsf{connect}), M(\mathsf{update})).$$

Der Beobachter sieht also die gesamte Markierung. Der Beobachter von Σ_5 soll auch eine Markierung von Σ_4 sehen. Dazu bilden wir von jeder Markierung von Σ_5 die

Stelle agents folgendermaßen ab: alle Agenten mit gleichem Level und Namen von der Stelle agents fassen wir zu einem Fragment zusammen:

$$f(M)(\mathsf{agents}) = \big[(V,E,n,l) \mid n \in A \wedge l \in \mathbb{N} \wedge V = \{x \mid (x,n,l) \in \mathsf{agents}_{(1,3,4)}\} \\ \wedge E = \bigcup\{E' \mid (E',n,l) \in \mathsf{agents}_{(2,3,4)}\}\big]$$

Die Stellen connect und update bleiben unverändert. Wir definieren den Beobachter von Σ_5 also als $\mathrm{Obs}_5(M) = (f(M)(\mathsf{agents}), M(\mathsf{connect}), M(\mathsf{update}))$.

Die beobachtbaren Abläufe von obs_5 des Ablaufs aus Abb. 7.15 sind genau die beobachtbaren Abläufe von obs_4 der beiden Abläufe aus Abb. 7.10 und Abb. 7.11 zusammen[7].

Das System Σ_5 ist mit den Beobachtern obs_4 und obs_5 eine Beobachterverfeinerung von Σ_4. Beide Systeme haben dieselben beobachtbaren Abläufe, obwohl sie unterschiedliche verteilte Abläufe haben.

Aus den bekannten Eigenschaften von Σ_4 und der Funktion f erhalten wir die Eigenschaften von Σ_5:

$$\Box\, \exists n,l : \mathsf{agents}_{(1,3,4)} = A \times [(n,l)] \\ \longrightarrow (\mathsf{agents}_{(1)}, \bigcup \mathsf{agents}_{(2)}) = \mathrm{mst}(A,N,w) \qquad (S_5)$$

$$\Diamond\, \exists n,l : \mathsf{agents}_{(1,3,4)} = A \times [(n,l)] \qquad (L_5)$$

Wir hatten in der Problemstellung des GHS-Algorithmus nicht nur gefordert, daß der minimale spannende Baum irgendwie berechnet wird, sondern so berechnet wird, daß jeder Agent seine Baumnachbarn kennt. Deshalb spezifizieren wir die Korrektheit von Algorithmus 5 mit den Formeln (S_5) und (L_5) zusammen mit der Forderung, daß sich jeder Agent nur Kanten merkt, die von ihm selbst ausgehen:

$$\Box\, \big(\mathsf{agent}(x,M,n,l) \wedge (x',y) \in M\big) \longrightarrow x = x' \qquad (A_5)$$

Die Gültigkeit von (A_5) folgt direkt aus der Definition von $F + (x,y)$.

7.6 Vereinigung in zwei nebenläufigen Schritten

Informelle Beschreibung

Bisher war die Vereinigung zweier Fragmente eine gemeinsame Aktion beider Fragmente. Wir werden diese Aktion nun zu zwei nebenläufig zueinander ausgeführten Aktionen der beiden Fragmente verfeinern.

Bei einer Vereinigung ändert sich für jeden Agenten der beiden Fragmente der Fragmentname und der Level wird um eins erhöht. Diese Veränderungen werden nun

[7] In Abb. 7.10 und Abb. 7.11 sind Abläufe von Σ_3 dargestellt. Durch Einfügen der Fragmentnamen erhalten wir daraus Abläufe von Σ_4

von beiden Fragmenten nebenläufig zueinander vorgenommen. Ein Fragment kann eine Vereinigungsaktion (also eine Umbenennung aller Agenten) ausführen, wenn es eine connect-Nachricht (x, y, l) mit gleichem Level wie seinem eigenen empfängt und selbst schon eine connect-Nachricht (y, x, l) über diese Kante verschickt hat. Das ist der Fall, wenn (x, y) bereits zur Kantenmenge des Fragments gehört.

Der Grund dafür ist der folgende: Schon seit Algorithmus 3 merkt sich jedes Fragment, daß es eine connect-Nachricht (y, x, l) an den Agenten y außerhalb des Fragments abgeschickt hat, indem es die Kante (x, y) in seine Kantenmenge aufnimmt. Solange diese Nachricht nicht angenommen ist, ändert sich der Level des Fragments nicht. Immer wenn eine ausgehende Kante zur Kantenmenge eines Fragments gehört, weiß das Fragment also, daß es über diese Kante eine connect-Nachricht mit seinem aktuellen Level abgeschickt hat. Wenn über diese Kante auch eine connect-Nachricht mit dem gleichen Level zurückkommt, weiß das Fragment, daß alle Voraussetzungen einer Vereinigung der beiden Fragmente gegeben sind, ohne noch einmal mit dem anderen Fragment zu kommunizieren. Diesen Umstand machen wir uns in Algorithmus 6 zunutze, indem wir den beiden Fragmenten ohne weitere Kommunikation miteinander erlauben, ihren Beitrag zur Vereinigung auszuführen. Die beiden Fragmente versehen unabhängig voneinander alle ihre Agenten mit neuem Fragmentnamen und neuem Level.

Das Petrinetzmodell

Die Vereinigung zweier Fragmente war bisher eine gemeinsame Aktion der beiden Fragmente, die in Σ_5 durch die Transition $t1$ modelliert ist. Diese Transition werden wir nun so verfeinern, daß eine Vereinigungsaktion im System Σ_6 zwei nebenläufig zueinander ausgeführten Aktionen der beiden vereinigenden Fragmente entspricht.

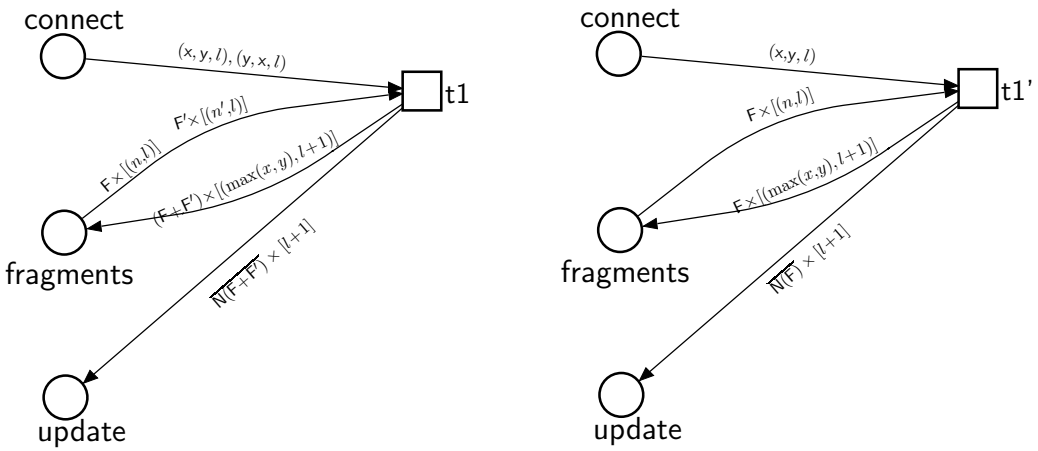


Abb. 7.16: Transition t1 und ein algebraisches Ersetzungsnetz N_E für t1

Ein Fragment mit Level l kann eine solche Aktion ausführen, wenn es eine connect-Nachricht (x, y, l) hat, der Agent x zum Fragment gehört und außerdem die Kante (x, y) schon als Baumkante ermittelt wurde. Damit ist sichergestellt, daß vom Fragment von x bereits eine connect-Nachricht (y, x, l) abgeschickt wurde. Die Aktion wird für beide Fragmente, die bei der Vereinigung mitmachen, gleichzeitig aktiviert,

nämlich genau dann, wenn das letzte der beiden Fragmente seine connect-Nachricht abgeschickt hat.

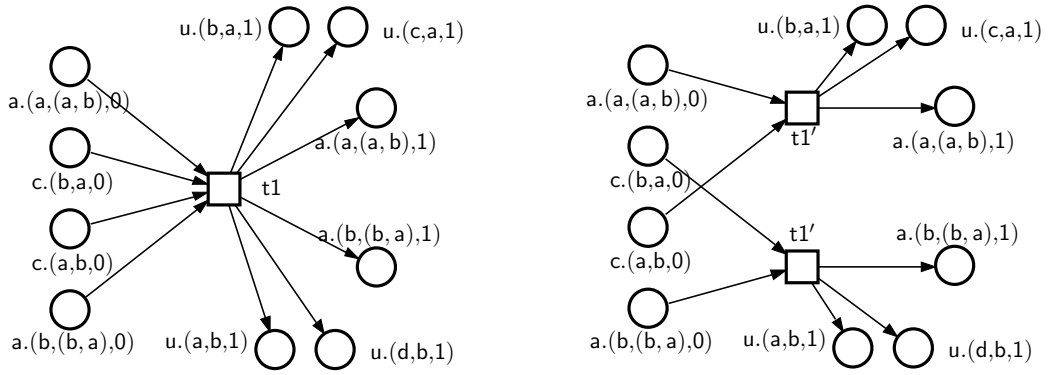


Abb. 7.17: Die Ersetzung von t1 in einem konkreten Schaltmodus

In Abbildung 7.16 wird die Transition $t1$ noch einmal als Ausschnitt von Σ_5 dargestellt. Daneben ist ein algebraisches Ersetzungsnetz für $t1$ dargestellt. Das Ersetzungsnetz hat zwar auch nur eine Transition, aber jede Aktion dieser Transition benötigt weniger Marken und gibt weniger Marken aus, als eine Aktion mit dem gleichen Modus in Σ_5. In Abbildung 7.17 haben wir das Auftreten der Transition $t1$ im konkreten Modus aus dem Ablauf 7.15 isoliert und daneben die Ersetzung für diesen Modus angegeben. Das ganze System Σ_6 ist in Abbildung 7.18 dargestellt.

Abbildung 7.19 zeigt einen Ablauf von Σ_6 für das Kommunikationsnetzwerk aus Abb. 7.9. Diesen Ablauf erhält man aus dem in Abb. 7.15 dargestellten Ablauf von Σ_5, indem man das Auftreten der Transition t1 wie in Abb. 7.17 ersetzt.

Korrektheit

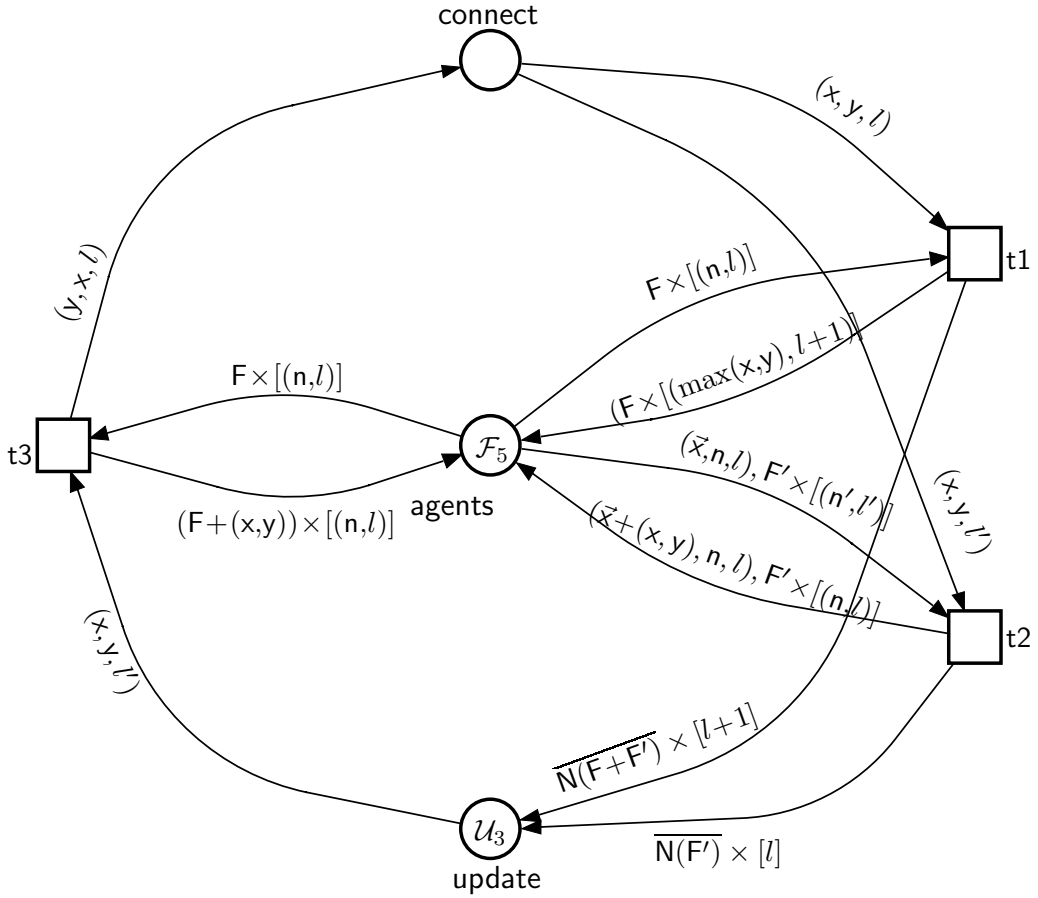
Die Korrektheit von Σ_6 spezifizieren wir genau wie die Korrektheit von Σ_5. Wir weisen die Korrektheit von Σ_6 nach, indem wir zeigen, daß das in Abb. 7.16 angegebene algebraische Ersetzungsnetz N_E eine Transitionsverfeinerung ist.

Zuerst stellen wir fest, daß N_E wirklich ein Ersetzungsnetz ist. Für jeden Modus β gilt: Der Ablauf von $(N_E, (t1, \beta)^-)$ endet im Zustand $(t1, \beta)^+$.

Wir teilen nun die Modi der Transition $t1$ in disjunkte Klassen ein. Wir bilden eine Klasse für je zwei verschiedene Agenten a, b und einen festen Level k und alle Modi der Transition $t1$ mit $[x = a, y = b, l = k]$ oder $[x = b, y = a, l = k]$. Wir betrachten das Ersetzungsnetz für diese Klasse von Modi, das wir kanonisch aus der Entfaltung des Ersetzungsnetzes erhalten. Für verschiedene Klassen von Modi sind die Ersetzungsnetze disjunkt. Nach Satz 3.7 brauchen wir nur zu beweisen, daß ein Ersetzungsnetz für eine Klasse von Modi eine Transitionsverfeinerung für diese Klasse ist.

Wir betrachten deshalb das System Σ_6', das wir aus Σ_5 erhalten, indem wir für einen festen Modus $[x = b, y = a, l = k]$ und $[x = a, y = b, l = k]$ die Transition $t1$ ersetzen und für alle anderen Modi der Transition beibehalten.

Wir stellen zuerst fest, daß die Abläufe des Ersetzungsnetzes im Modus $[x = b, y = a, l = k]$ und im Modus $[x = a, y = b, l = k]$ identisch sind. Außerdem kommt in



Aktivierungsbedingung

t1 : $x \in F \wedge (x,y) \in F \wedge \forall z, z' : (z,z') \in F \rightarrow z' \in F \vee z' = y$

Alle anderen Aktivierungsbedingungen, Sorten und Konstanten wie in Σ_5.

Abb. 7.18: Σ_6

jedem Ablauf von Σ_6' die Transition $t1'$ im Modus $[x = a, y = b, l = k]$ höchstens einmal vor, denn danach erhöhen sich die Level von a und b.

Wir müssen für jeden Ablauf von Σ_6' beweisen, daß es zwei Schnitte gibt, zwischen denen genau die beiden Aktionen des Ersetzungsnetzes zum Modus $[x = a, y = b, l = k]$ liegen. Dazu reicht es zu zeigen: Wenn die Transition t1′ im Modus (a, b, k) auftritt, dann tritt nebenläufig dazu die Transition t1′ im Modus (b, a, k) auf.

Sei also ρ ein Ablauf von Σ_6'. Angenommen, die Transition t1′ tritt im Modus (a, b, k) auf. Da der Ablauf bis zum Auftreten dieser Aktion identisch mit einem Anfangsstück eines Ablaufs von Σ_5 ist, können wir schließen, daß die Transition t1′ für beide Modi gleichzeitig aktiviert wurde. Die einzigen, dazu in Konflikt stehenden Aktionen sind Beitritte, an dem nur ein Agent des Fragments beteiligt ist. Nach einem solchen Beitritt hat sich das Fragment zwar vergrößert, aber t1′ ist immer noch für das Fragment aktiviert. Wenn die Transition t1′ für den Modus

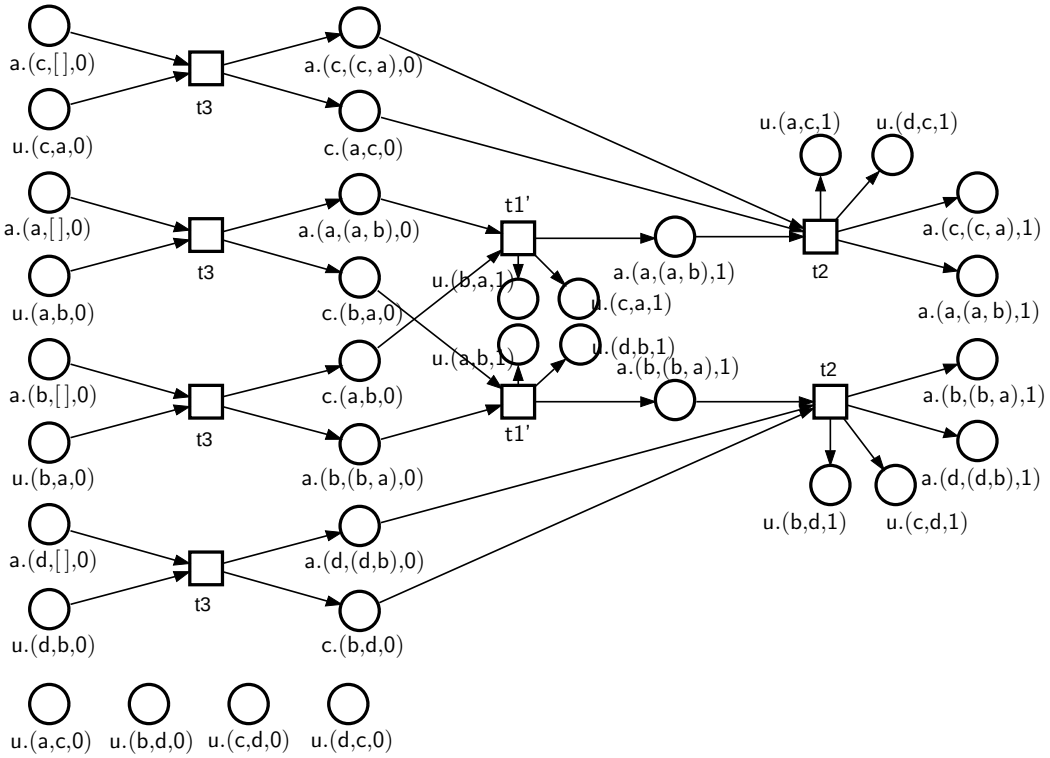


Abb. 7.19: Ein verteilter Ablauf von Σ_6

$[x = b, y = a, l = k]$ aktiviert ist, tritt sie auch in diesem Modus im Ablauf auf. Es ist aber nicht deterministisch, welche Agenten genau zum Fragment des Agenten b gehören.

Da in einem Ablauf des Ersetzungsnetzes die beiden Aktionen der Transition $\mathsf{t1}'$ im Modus $[x = b, y = a, l = k]$ und im Modus $[x = a, y = b, l = k]$ nebenläufig zueinander auftreten, müssen wir nun zeigen, daß sie auch in ρ zueinander nebenläufig sind. Die Vor- und Nachbereiche der beiden Aktionen sind disjunkt, so daß sie nicht direkt hintereinander auftreten können. Es muß also mindestens eine Aktion zwischen den Aktionen $(\mathsf{t1}', [x = a, y = b, l = k])$ und $(\mathsf{t1}', [x = b, y = a, l = k])$ geben. Wir können weiter zeigen, daß mindestens eine $\mathsf{t3}$-Aktion zwischen den beiden Aktionen liegt.

Wir betrachten nun zu jeder Aktion im Ablauf die Menge aller Agenten, die an dieser Aktion beteiligt sind. Wenn für ein Fragment die Aktion zum Abschicken einer connect-Nachricht aktiviert ist, sind an allen Aktionen im Ablauf, die kausal vor der Marke dieses Fragments liegen, höchstens Agenten dieses Fragments beteiligt. Wir können damit zeigen, daß an der $\mathsf{t3}$-Aktion zwischen $(\mathsf{t1}', [x = a, y = b, l = k])$ und $(\mathsf{t1}', [x = b, y = a, l = k])$ bereits alle Agenten von des Fragments des Agenten a beteiligt sind. Wenn dieses vergrößerte Fragment nun dem Fragment des Agenten b beitritt, dann gibt es einen Kreis im minimalen spannenden Baum, da a und b bereits über eine Baumkante verbunden sind. Dies ist ein Widerspruch.

Bei dieser Argumentation haben wir immer berücksichtigt, daß $\mathsf{t1}$ nur im Modus $[x = b, y = a, l = k]$ und im Modus $[x = a, y = b, l = k]$ ersetzt wurde. In allen anderen Modi schaltet $\mathsf{t1}$ wie vorher. Dadurch konnten wir alle Eigenschaften von Σ_5 benutzen, denn bis zum Auftreten von $(\mathsf{t1}', [x = a, y = b, l = k])$ und $(\mathsf{t1}', [x =$

$b, y = a, l = k]$) ist der Ablauf von Σ'_6 identisch mit dem Präfix eines Ablaufs von Σ_5.

Nach Satz 3.7 ist das Ersetzungsnetz auch eine Transitionsverfeinerung, wenn wir alle Modi simultan ersetzen.

7.7 Erste Kommunikation im Fragment – Verteilte Umbenennung

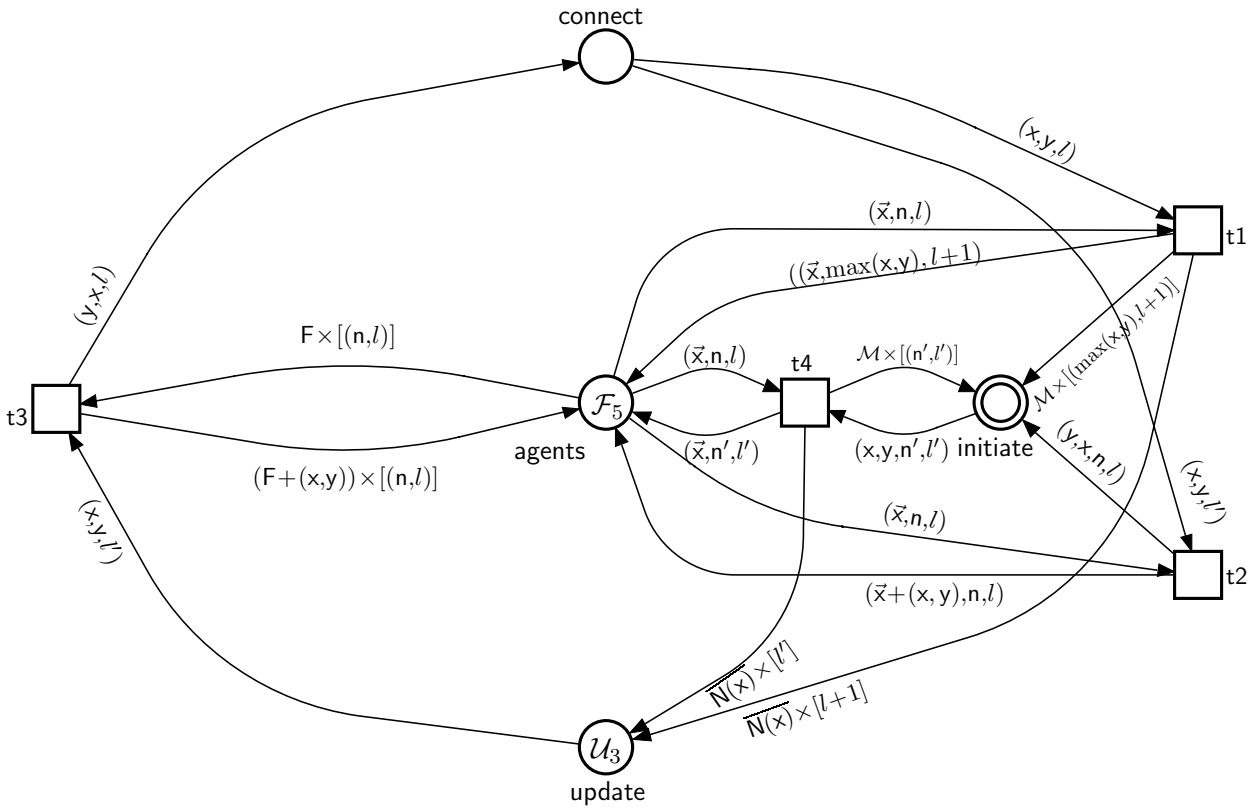
Informelle Beschreibung

Die Kommunikation zwischen den Fragmenten haben wir in den letzten Abschnitten beschrieben. Im weiteren wird es darum gehen, wie die Agenten innerhalb eines Fragments miteinander kommunizieren. In diesem Verfeinerungsschritt ersetzen wir die Transitionen `t1` und `t2` durch einen verteilten Algorithmus, in dem die einzelnen Agenten nur über Nachrichtenaustausch kommunizieren und keine synchronen Aktionen mehr haben.

Sehen wir uns noch einmal an, was im letzten Abschnitt in Algorithmus 6 bei einer Vereinigung oder einem Beitritt passiert: Alle Agenten eines Fragments bekommen gleichzeitig einen neuen Namen und einen neuen Level. Bei einer Vereinigung oder einem Beitritt gibt es immer eine ausgezeichnete Kante (x, y), über die die beiden beteiligten Fragmente sich zu einem neuen Fragment verbinden. In Algorithmus 7 sind an einer Verbindung (Vereinigung oder Beitritt) zweier Fragmente zuerst nur die beiden Agenten x und y beteiligt. Diese Agenten initiieren dann eine Nachrichtenwelle durch das neue Fragment, die allen Agenten den neuen Namen und Level mitteilt.

Als Kommunikationsstruktur innerhalb eines Fragments werden nicht alle Kommunikationskanäle benutzt, sondern nur die, die bereits als zum minimalen spannenden Baum gehörig berechnet wurden. Diese bilden innerhalb des Fragments auch einen Baum. Ein Agent, der eine Nachricht mit neuem Namen und Level erhält, schickt sie an alle seine Baumnachbarn außer dem Absender weiter.

Den hier benutzten verteilten Algorithmus kann man auch als die erste Phase des Echo-Algorithmus (auch bekannt als Segalls's PIF-Protokoll [Seg83]) ansehen. In dieser Phase wird ausgehend von einem ausgezeichneten Agenten (dem Initiator) eine Information allen Agenten des Netzwerks bekanntgemacht. Der Initiator schickt die Information an alle seine Nachbarn. Jeder Agent x, der die Information von einem Agenten y erhält, schickt sie an alle seine Nachbarn außer y weiter. In der zweiten Phase des Echo-Algorithmus werden Antworten von allen Agenten des Netzwerks eingesammelt. Die zweite Phase werden wir erst in einem späteren Verfeinerungsschritt in unseren Algorithmus einbauen. Dieser Algorithmus ist jedoch für beliebige zusammenhängende Netzwerke definiert. Das Wichtigste ist, daß während des Algorithmus ein spannender Baum im Netzwerk konstruiert wird. Da wir von Anfang an einen Baum vorgegeben haben, nämlich den Teilbaum des minimalen spannenden Baumes, der das Fragment ausmacht, ist der Algorithmus fast trivial.



Sorten

$\mathcal{M}$: 2^N

initiate: $N \times A \times \mathbb{N}$

Aktivierungsbedingungen

t1 : $\vec{x} = (x, M) \ \wedge \ (x, y) \in M \ \wedge \ \mathcal{M} = \overline{M} \setminus \{(y, x)\}$

t2 : $l' < l$

t4 : $\vec{x} = (x, M) \ \wedge \ \mathcal{M} = \overline{M} \setminus \{(y, x)\}$

Alle anderen Aktivierungsbedingungen, Sorten und Konstanten wie in Σ_6.

Abb. 7.20: Σ_7

Das Petrinetzmodell

Bei der Vereinigung wird ausgehend von den beiden Agenten an der Vereinigungskante die Nachricht über den neuen Namen und Level des Fragments an alle Agenten des Fragments weitergegeben. Ein Agent x startet eine solche Welle, wenn er eine connect-Nachricht von einem Agenten y erhält und der Agent y bereits zu seinen Baumnachbarn gehört. In diesem Fall weiß x nämlich, daß er auch schon eine connect-Nachricht an y geschickt hat. Dann schickt x eine initiate-Nachricht, die den neuen Namen (Maximum von x und y) und neuen Level (Level von x plus 1) enthält an alle seine Baumnachbarn außer y (Transition t1). Jeder Agent x, der eine

initiate-Nachricht von einem Agenten y bekommt, schickt die Nachricht an alle seine Baumnachbarn außer y weiter (Transition t4).

Ähnlich ist es bei einem Beitritt. Hier ist der Initiator der Nachrichtenwelle der Agent mit dem größeren Level l, der das Fragment mit dem kleineren Level l' beitreten läßt: Ein Agent x, der eine connect-Nachricht mit kleinerem Level von y empfängt, merkt sich y als Baumnachbar und schickt y eine initiate-Nachricht, die Namen und Level seines Fragments enthält (Transition t2). Diese Nachricht wird dann im Fragment mit Level l', wie bei der Vereinigung beschrieben, weiterverbreitet (Transition t4).

Die Stelle für die initiate-Nachrichten ist eine FIFO-Stelle (vgl. Abschnitt 4.3). Im Algorithmus 7 kann es passieren, daß ein Agent mehrere initiate-Nachrichten von demselben Nachbarn hat. Es ist wichtig, daß der Agent die Nachrichten in derselben Reihenfolge empfängt, wie sie abgeschickt wurden. Beide Nachrichten enthalten einen neuen Namen und Level für den Agenten. Wenn der Agent die aktuelle Nachricht zuerst bearbeitet und sich später auf Grund einer veralteten Nachricht umbenennt und diese auch noch weiterschickt, ist die Benennung des Fragments inkonsistent und der Algorithmus kann verklemmen.

Korrektheit

Die hier beschriebene Ersetzung der Transitionen t1 und t2 durch die erste Phase des Echo-Algorithmus ist eine simultane algebraische Transitionsverfeinerung.

Dies folgt ähnlich wie im letzten Verfeinerungsschritt. Wenn eine Nachrichtenwelle einmal begonnen hat, wird sie auch zu Ende geführt, denn jeder Agent, für den die Transition t4 aktiviert wird, wird diese Aktion irgendwann schalten. Die einzigen Aktionen, die für den Agenten damit in Konflikt stehen, sind Beitritte (t2). Nach einem Beitritt hat der Agent einen Nachbarn mehr, aber die Transition t4 ist immer noch für ihn aktiviert.

Jede Nachrichtenwelle kann durch zwei Schnitte aus dem Ablauf herausgeschnitten werden. Auch dies folgt mit einem ähnlichen Argument wie im letzten Verfeinerungsschritt: Die Agenten im System kommunizieren ausschließlich über die Kanten des minimalen spannenden Baumes miteinander (d.h. jede Nachricht die empfangen wird, ist eine Nachricht über eine Baumkante, update-Nachrichten werden auch über andere Kanten versendet, werden aber nur empfangen, wenn sie über Kante des minimalen spannenden Baumes versendet wurden). Jeder Agent führt in jeder Nachrichtenwelle höchstens eine Aktion aus. Man kann ähnlich wie im letzten Schritt folgern, daß es einen Kreis im minimalen spannenden Baum gibt, wenn es die beiden Schnitte nicht gibt, um eine Nachrichtenwelle herauszuschneiden. Die Eigenschaften (S) und (L) von Σ_6 bleiben nach Bemerkung 5.12 erhalten. Damit ist auch Σ_7 korrekt.

7.8 Die abstrakte Suche nach der kleinsten Kante

Informelle Beschreibung

Die verteilte Suche nach der kleinsten aus einem Fragment herausführenden Kante ist der komplizierteste Teil des GHS-Algorithmus. In den vier nächsten Verfeine-

rungsschritten werden wir diese Suche beschreiben.

Die Suche nach der kleinsten aus dem Fragment herausführenden Kante ist bisher noch nicht explizit modelliert, sondern in der Aktivierungsfunktion von Transition t3 ($(x, y) = \mathrm{mo}(F)$) versteckt. Bevor wir die lokale Suche jedes einzelnen Agenten und die Kommunikation darüber im Fragment beschreiben, führen wir in diesen Schritt eine neue Transition ein, die die Suche abstrakt als gemeinsame Aktion aller Agenten eines Fragments modelliert. Wir beschreiben dazu Suchzustände von Agenten, d.h. jeder Agent erhält eine neue Variable s, die formal angibt, ob er sich gerade an der Suche beteiligt ($s = \mathsf{find}$) oder nicht ($s = \mathsf{found}$).

Wenn sich zwei Fragmente vereinigen, gehen alle Agenten in den Zustand find über, das bedeutet: Die Suche nach der kleinsten Kante beginnt. Wenn ein Fragment einem Fragment mit größerem Level beitritt, übernimmt das beitretende Fragment nicht nur Namen und Level, sondern auch den Suchzustand. Wenn das Fragment mit größerem Level gerade sucht (Zustand find), dann sucht das beitretende Fragment mit, wenn das Fragment die Suche bereits abgeschlossen hat, dann nimmt das beitretende Fragment gleich den Zustand found an. Wenn eine connect-Nachricht abgeschickt wird, muß die Suche abgeschlossen sein, d.h. alle Agenten des Fragments befinden sich im Zustand found.

Das Petrinetzmodell

In diesem Modell benutzen wir die neue Sorte $\mathbb{S}\mathrm{tate} = \{\mathsf{find}, \mathsf{found}\}$. Eine Marke auf der Stelle agents hat nun die Form (x, M, n, l, s). Dabei ist

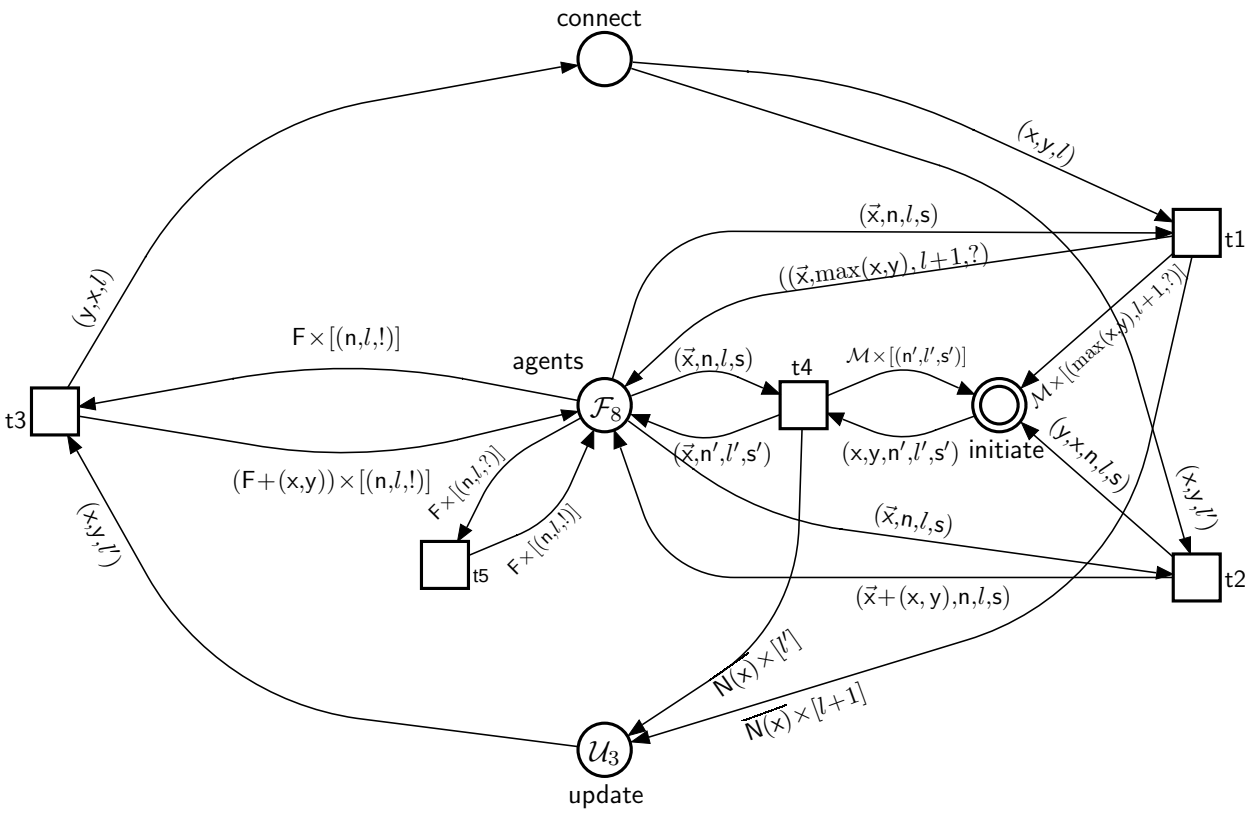
- x der Name des Agenten,
- M die Menge der von x ausgehenden Kanten die schon als Kanten des minimalen spannenden Baumes ermittelt wurden,
- n der Name des Fragments von x,
- l der Level des Fragments von x,
- s der Suchzustand find oder found.

Beim Schalten der neuen Transition t5 gehen alle Agenten eines Fragments gemeinsam vom Zustand find in den Zustand found über. Diese Transition beschreibt die Suche nur abstrakt, denn der Zustandsübergang von find nach found hat in diesem Algorithmus noch keine Semantik, d.h. wir modellieren nicht, *wie* gesucht wird, und wir halten noch nicht einmal ein Ergebnis fest.

Wir erklären nun den Einfluß des Suchzustandes auf die anderen Transitionen. Am Anfang sind alle Agenten im Zustand found.

Transition t3: Die Transition t3 ist nur aktiviert, wenn alle Agenten des Fragments im Zustand found sind, also die Suche abgeschlossen haben. Wenn die Transition t3 schaltet, bleiben alle Agenten im Zustand found.

Transition t1: Beim Schalten der Transition t1 geht der Agent in den Zustand find über, denn es bildet sich ein neues Fragment und die Suche beginnt. Den Zustand find schickt er in der initiate-Nachricht an seine Baumnachbarn weiter.



Sorten		Konstanten
s :	$\mathbb{S}\text{tate}$	$\mathcal{F}_8 = \mathcal{F}_5 \times \{\text{found}\}$
agents :	$\mathbb{A}\text{gent} \times A \times \mathbb{N} \times \mathbb{S}\text{tate}$	! = found
initiate :	$N \times A \times \mathbb{N} \times \mathbb{S}\text{tate}$	? = find

Aktivierungsbedingungen

t5 : $\quad \forall z, z' : \big((z, z') \in F \rightarrow z' \in F\big)$

Alle anderen Aktivierungsbedingungen, Sorten und Konstanten wie in Σ_7.

Abb. 7.21: Σ_8

Transition t2: Beim Schalten der Transition t2 ändert ein Agent seinen Zustand nicht, aber er gibt seinen Zustand (egal ob find oder found) in der initiate-Nachricht weiter.

Transition t4: Ein Agent, der eine initiate-Nachricht empfängt, ändert nun nicht nur seinen Namen und Level, sondern nimmt auch den Zustand an, der in der Nachricht angegeben ist und schickt ihn weiter.

Transition t5: Wir führen eine neue Transition t5 ein, die abstrakt die Suche modelliert. Die Transition ist aktiviert, wenn alle Agenten eines Fragments im Zustand find sind. Beim Schalten gehen alle Agenten eines Fragments gemeinsam vom Zustand find in den Zustand found über.

Korrektheit

Die Einführung des Zustands ist eine Beobachterverfeinerung bzgl. folgender Beobachter: Obs_7, der Beobachter von Σ_7 sieht die gesamte Markierung von Σ_7, Obs_8, der Beobachter von Σ_8 sieht auch alle Stellen des Systems Σ_8, aber er blendet überall den Suchzustand aus, so daß er auch eine Markierung von Σ_7 sieht.

Die neue Transition t5 ändert nichts an den alten Daten und die anderen Transitionen verändern die alten Daten genauso wie in Algorithmus 7. Die einzige Transition, an der diese neue Variable zusätzlich synchronisiert, ist Transition t3. Es ist eine zusätzliche Aktivierungsbedingung für diese Transition, daß alle Agenten den Zustand found haben.

Wir können zeigen, daß die Systeme Σ_7 und Σ_8 mit den Beobachtern Obs_7 und Obs_8 die Kriterien von Satz 5.22 erfüllen.

Jeder Ablauf von Σ_7 ist endlich. Die einzigen nicht beobachtbaren Aktionen von Σ_8 sind die Aktionen von t5. Die neue Transition t5 ändert nur Suchzustände. Wenn diese Transition schaltet, ist das ein Stotterschritt bzgl. der beobachtbaren Markierung von Σ_7. Solche Stotterschritte können nicht unendlich oft hintereinander vorkommen, da das Fragment erst wieder in den Zustand find übergehen muß, bevor die Transition erneut schalten kann. Da wir nun zeigen, daß alle anderen Transition von Σ_7 simuliert werden, ist auch jeder Ablauf von Σ_8 endlich.

- Die beobachtbaren Anfangszustände der Systeme stimmen überein.

- Wir betrachten nun einen beliebigen erreichbaren Zustand M von Σ_8. Für die Transitionen t1, ... ,t4 gilt: Wenn eine Transition in M_8 in Σ_8 aktiviert ist, dann ist sie in der Markierung M_7 mit $\text{obs}_7(M_7) = \text{obs}_8(M'_8)$ auch in Σ_7 aktiviert.

- Wir müssen noch zeigen, daß kein Ablauf von Σ_8 zu früh abbricht und dadurch die Lebendigkeitseigenschaft verletzt. Dazu zeigen wir, daß immer, wenn t3 in einer erreichbaren Markierung M_7 in Σ_7 aktiviert ist, dann ist t3 oder t5 in jeder erreichbaren Markierung M_8 mit $\text{obs}_7(M_7) = \text{obs}_8(M'_8)$ in Σ_8 aktiviert: Alle Agenten, die gleiche Fragmentnamen und gleiche Level haben, sind auch im gleichen Zustand. Diese Invariante ist leicht an jeder Transition nachprüfbar. Wenn also t3 bis auf die zusätzliche Zustandsbedingung aktiviert ist, dann müssen alle Agenten im Zustand find sein. In diesem Fall kann t5 schalten. Nach dem Schalten von t5 sind alle Agenten im Zustand found und t3 kann schalten.

Wir haben gezeigt, daß Σ_8 eine Beobachterverfeinerung bzgl. der Beobachter Obs_7 und Obs_8 von Σ_7 ist. Da Obs_7 alles und Obs_8 alles außer dem Suchzustand sieht, ist auch dieser Algorithmus korrekt bzgl. der Eigenschaften $(S_5), (L_5)$ und (A_5).

7.9 Neue Daten

Informelle Beschreibung

Zur Kommunikation innerhalb des Fragments während der Suche nach der kleinsten aus dem Fragment herausführenden Kante benötigt jeder Agent neue Variablen, in denen er vorübergehend Information speichern kann. In diesem Verfeinerungsschritt führen wir für jeden Agenten eine Variable ein. Diese neue Variable wird erst im nächsten Verfeinerungsschritt benutzt. Der Verfeinerungsschritt ist einfach und rein technisch.

Von nun an hat jeder Agent eine neue Variable, in der er speichert, von wem er die letzte initiate-Nachricht erhalten hat. Wir nennen diese Variable den Vater des Agenten. Am Anfang des Algorithmus ist jeder Agent sein eigener Vater. Ein Agent, der die Vereinigungstransition t1 schaltet, merkt sich den neuen Namen seines Fragments als Vater. Das ist entweder er selbst oder der Agent, von dem er die connect Nachricht erhalten hat. Ein Agent, der eine initiate-Nachricht empfängt, merkt sich den Absender der Nachricht als Vater.

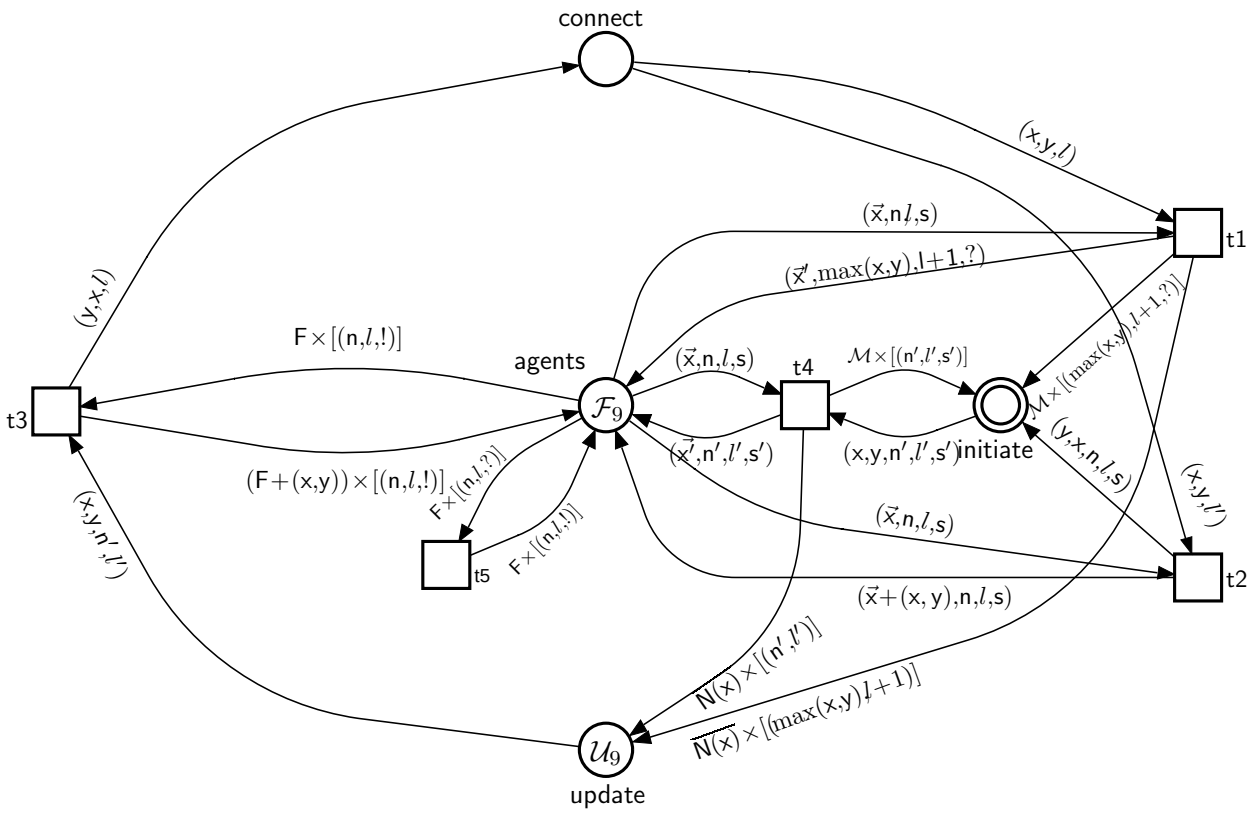
Bei jeder Umbenennung eines Fragments mit neuem Namen n entsteht durch die Vaterrelation ein Wurzelbaum mit Wurzel n im Fragment. Der Vater wird im nächsten Verfeinerungsschritt benötigt, wenn wir die zweite Phase des Echo-Algorithmus modellieren. Dabei werden über den Vaterbaum die Antworten der Agenten eines Fragments eingesammelt. Die Welle von Antworten endet beim Namensgeber des Fragments.

Eine weitere kleine Datenänderung, die wir erst im übernächsten Verfeinerungsschritt benutzen, ist die folgende: Von nun an schickt jeder Agent in jeder update-Nachricht nicht nur seinen Level, sondern auch seinen Fragmentnamen mit. Mit dieser zusätzlichen Information kann ein Agent in einem späteren Verfeinerungsschritt entscheiden, ob ein Nachbar bereits zum selben Fragment gehört wie er selbst oder nicht.

Das Petrinetzmodell

Eine Marke auf der Stelle Agents hat nun die Form (x, M, v, n, l, s). Dabei ist

- x der Name des Agenten,
- M die Menge der von x ausgehenden Kanten die schon als Kanten des minimalen spannenden Baumes ermittelt wurden,
- v der Vater von x, also der Agent, von dem x die letzte initiate-Nachricht erhalten hat,
- n der Name des Fragments von x,
- l der Level des Fragments von x,
- s der Suchzustand find oder found.



Sorten		Konstanten	
$\vec{x}, \vec{x}'$:	$\mathbb{A}\text{gent} \times A$	$\mathcal{F}_9$	$= \{(a, \emptyset, a, a, 0, !) \mid a \in A\}$
v :	A	$\mathcal{U}_9$	$= \{(y, a, a, 0) \mid (a, b) \in N\}$
agents :	$\mathbb{A}\text{gent} \times A \times A \times \mathbb{N} \times \mathbb{S}\text{tate}$		
update :	$N \times A \times \mathbb{N}$		

Aktivierungsbedingungen

t1 : $\vec{x} = (x, M, v) \ \wedge \ \mathcal{M} = \overline{M} \setminus \{(y, x)\} \ \wedge \ \vec{x'} = (x, M, \max(x, y))$

t4 : $\vec{x} = (x, M, v) \ \wedge \ \mathcal{M} = \overline{M} \setminus \{(y, x)\} \ \wedge \ \vec{x'} = (x, M, y)$

Alle anderen Aktivierungsbedingungen, Sorten und Konstanten wie in Σ_8.

Abb. 7.22: Σ_9

Um die Kanteninschriften übersichtlich zu halten, definieren wir das Speichern des Vaters in den Aktivierungsbedingungen von t1 und t4, statt direkt in den Kanteninschriften. Wir definieren die Sorte der Variable $\vec{x}$ neu, so daß der Vater als letzte Komponente in dieser Variable enthalten ist.

Korrektheit

Dieser Verfeinerungsschritt ist eine konservative Datenerweiterung. Die neuen Variablen synchronisieren an keiner Transition. Die Korrektheit bleibt erhalten.

7.10 Die Rückwelle des Echo-Algorithmus

Informelle Beschreibung

Wir verfeinern nun den Übergang eines Fragments vom Zustand find in den Zustand found zu einem nachrichtenbasierten Algorithmus, nämlich der zweiten Phase des Echo-Algorithmus. In Algorithmus 10 gehen nicht mehr alle Agenten gemeinsam vom Zustand find in den Zustand found über, sondern jeder Agent einzeln.

Seit dem letzten Verfeinerungsschritt merkt sich ein Agent, der in den Zustand find übergeht, seinen Vater. Ein Agent geht wieder in den Zustand found, wenn er von jedem seiner Baumnachbarn außer seinem Vater eine report-Nachricht erhalten hat. Die Agenten, die Blattknoten[8] im Fragment sind, beginnen mit diesem Übergang, denn sie brauchen nicht auf eine Nachricht zu warten. Der letzte Agent des Fragments, der diesen Übergang macht, ist der Namensgeber des Fragments, denn er ist die Wurzel des Vaterbaumes.

Das Petrinetzmodell

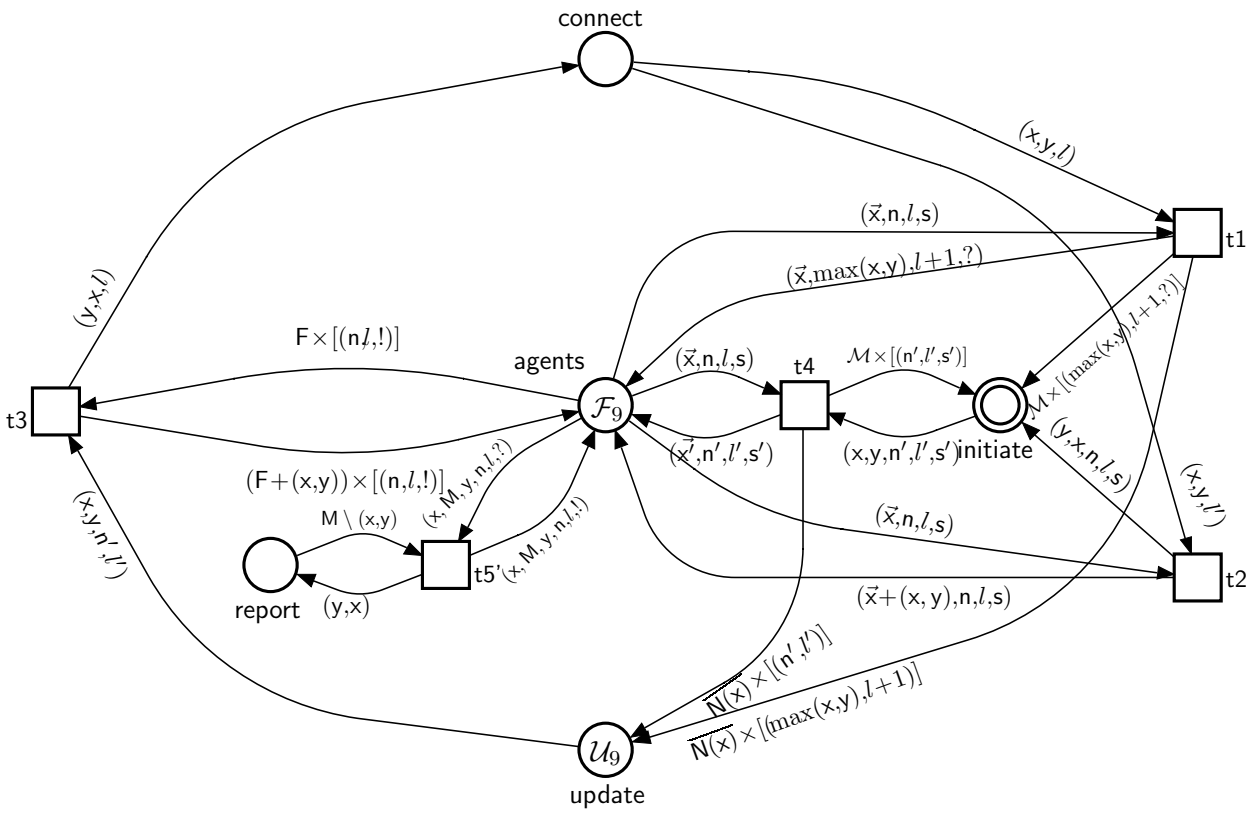
In diesem Modell haben wir eine neue Stelle report für die report-Nachrichten. Die Transition t5 wurde durch die Transition t5′ ersetzt. t5′ schaltet, wenn ein Agent im Zustand find eine report-Nachricht von jedem seiner Baumnachbarn außer seinem Vater bekommen hat. Der Agent geht dann in den Zustand found über und schickt eine report-Nachricht an seinen Vater weiter. Ein Blattknoten im Fragment schaltet diese Transition, ohne eine report-Nachricht zu erhalten, denn er hat keinen Baumnachbarn außer seinem Vater.

Eine report-Nachricht enthält in diesem Algorithmus keine Information (außer Empfänger und Absender). Im nächsten Verfeinerungsschritt werden die report-Nachrichten zum Einsammeln der Suchergebnisse der lokalen Suche der Agenten verwendet.

Korrektheit

Dieser Verfeinerungsschritt ist eine Transitionsverfeinerung der Transition t5 durch die Transition t5′. In jedem Ablauf von Σ_9 wird jedes Auftreten von t5 durch die zweite Phase des Echo-Algorithmus im Fragment ersetzt, also durch einen Teilablauf, in dem die Transition t5′ für jeden Agenten des Fragments genau einmal vorkommt. Die Transitionen der Agenten sind dabei genauso geordnet wie die Agenten in der Vaterrelation, d.h. wenn es im Vaterbaum einen Weg vom Agenten x zu y gibt, dann tritt die Transition von x vor der Transition von y auf. Als letzte tritt die

[8] Diese Agenten haben keinen Baumnachbarn außer ihrem Vater.



Sorten

M :	2^N
report :	N

Aktivierungsbedingungen

t5′ : true

Alle anderen Aktivierungsbedingungen, Sorten und Konstanten wie in Σ_9.

Abb. 7.23: Σ_{10}

Transition des Namensgebers des Fragment auf, denn dieser Agent ist die Wurzel des Vaterbaumes.

Die Korrektheit des Verfeinerungsschrittes wird ähnlich wie die Korrektheit des Schrittes von Σ_6 nach Σ_7 bewiesen.

7.11 Die Suche nach der kleinsten Kante

Informelle Beschreibung

In diesem Verfeinerungsschritt modellieren wir die Suche nach der kleinsten aus einem Fragment herausführenden Kante als verteilten Algorithmus. Der Algorithmus

besteht aus zwei Teilen: Jeder Agent sucht zuerst seine kleinste adjazente Kante, die aus dem Fragment herausführt (lokale Suche). Dann werden die Gewichte dieser Kanten innerhalb des Fragments verglichen, um die kleinste aus dem Fragment herausführende Kante zu bestimmen (globale Suche).

Die lokale Suche

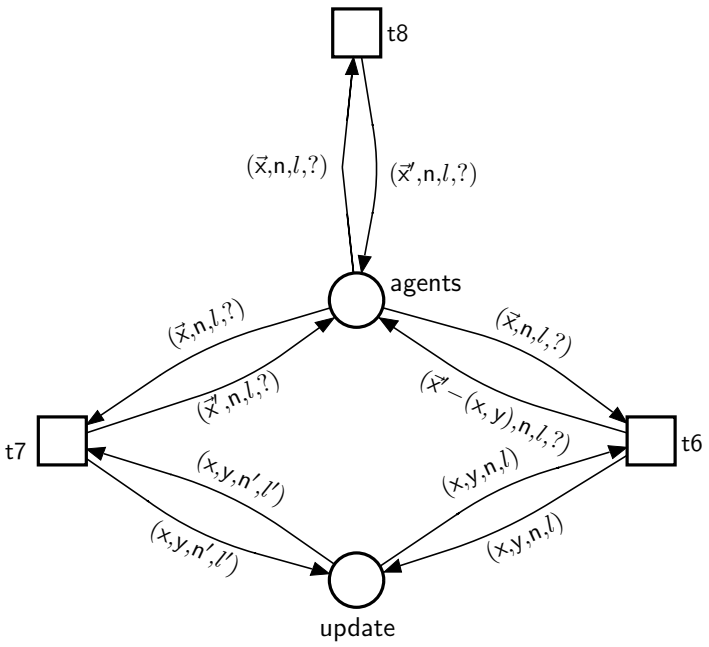
Wie kann ein Agent, der zur Suche aufgefordert ist (s = find) entscheiden, ob eine adjazente Kante aus dem Fragment herausführt oder nicht? Er schaut sich die update-Nachrichten von den Nachbarn an, die noch keine Baumnachbarn sind. Wenn er von einem dieser Nachbarn eine update-Nachricht mit seinem eigenen Fragmentnamen und Level erhalten hat, dann weiß er, daß die Kante zu diesem Agenten nicht aus dem Fragment herausführt. Hat er eine Nachricht mit einem kleineren Level als seinem eigenen, dann weiß er nichts, denn es könnte sein, daß der Absender der Nachricht bereits zu seinem Fragment gehört, aber die initiate-Nachricht mit dem neuen Namen und Level noch nicht empfangen hat. Ein Agent im Zustand find kann aber zweifelsfrei sagen, daß eine Kante aus dem Fragment herausführt, wenn er über diese Kante eine update-Nachricht empfangen hat, deren Level mindestens so groß ist wie sein eigener und deren Fragmentname von seinem verschieden ist.

Damit ein Agent nicht bei jeder Suche alle Kanten neu testen muß, erhält jeder Agent eine neue Variable B (wie "Basiskanten"), die ihm die Menge aller adjazenten Kanten angibt, über die er noch nicht weiß, ob sie zum minimalen spannenden Baum gehören oder nicht. Am Anfang sind alle seine adjazenten Kanten in dieser Menge. Wenn für den Agenten eine neue Baumkante ermittelt wurde (Transition t2 oder t3), dann entfernt er die Kante aus der Menge B.

Wenn ein Agent zur Suche aufgefordert wird, dann betrachtet er die Kante mit dem kleinsten Gewicht aus B und wartet so lange, bis er eine update-Nachricht über diese Kante bekommt, deren Level mindestens so groß ist, wie sein eigener. Wenn in der Nachricht sein eigener Fragmentname steht, dann entfernt er die Kante aus B. Falls die Menge B leer ist, beendet er die Suche, denn dann gibt es keine adjazente Kante, die aus dem Fragment herausführt. Ansonsten geht er zur nächstgrößeren Kante aus B über. Wenn in der update-Nachricht mit mindestens gleichgroßem Level ein anderer Fragmentname steht, merkt er sich diese Kante als kleinste adjazente aus dem Fragment herausführende (in einer neuen Variable b wie "beste Kante") und beendet die lokale Suche.

Mit den beiden neuen Variablen hat eine Marke auf der Stelle agents nun die Form (x, M, B, v, b, n, l, s). Dabei ist

- x der Name des Agenten,
- M die Menge der von x ausgehenden Kanten die schon als Kanten des minimalen spannenden Baumes ermittelt wurden,
- B die Menge der von x ausgehenden Kanten von denen er noch nicht weiß, ob sie zum minimalen spannenden Baumes gehören,
- v der Vater von x, also der Agent, von dem x die letzte initiate-Nachricht erhalten hat,



Sorten

B : 2^N

b : $A \cup \{\perp, \text{nil}\}$

Aktivierungsbedingungen

Es sei immer $\vec{x} = (x, M, B, v, b)$ und $\vec{x}' = (x, M', B', v', b')$
und falls nicht anders angegeben $M = M'$, $B = B'$ u.s.w.

t6 : $b = \perp \ \wedge \ (x, y) \in B$

t7 : $b = \perp \ \wedge \ l \leq l' \ \wedge \ n \neq n' \ \wedge \ w(x, y) = \min\{w(x, z) \mid (x, z) \in B\} \ \wedge \ b' = y$

t8 : $b = \perp \ \wedge \ B = \emptyset \ \wedge \ b' = \text{nil}$

Alle anderen Sorten und Konstanten wie in Σ_{10}.

Abb. 7.24: Die lokale Suche

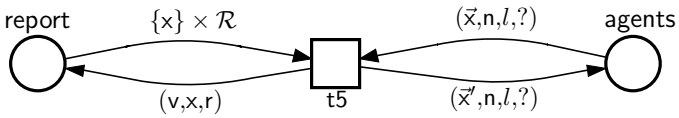
- b die kleinste von x aus dem Fragment herausführende Kante, bzw. der Sonderwert $\perp$, wenn diese Kante noch nicht ermittelt wurde oder der Wert nil, wenn keine solche Kante existiert,
- n der Name des Fragments von x,
- l der Level des Fragments von x,
- s der Suchzustand find oder found.

Wir verändern die Transitionen t1 und t4 aus Σ_{10} in Algorithmus 11 so, daß jeder Agent, der seinen Zustand in find verändert, seine Variable b auf $\perp$ setzt (dargestellt im gesamten Modell in Abb. 7.26). Damit löscht er einen alten Wert und weiß, daß er die lokale Suche noch nicht abgeschlossen hat.

Durch Transition t6 wird eine Kante aus B entfernt, weil sie nicht aus dem Fragment herausführt (und deshalb nicht zum minimalen spannenden Baum gehört). Wenn ein Agent seine kleinste herausführende Kante gefunden hat, speichert er den Namen des Nachbarn an dieser Kante (Transition t7). Wenn ein Agent keine adjazente, aus dem Fragment herausführende Kante hat, dann schließt er seine lokale Suche ab, indem er den Wert für b (beste Kante) auf nil setzt (Transition t8).

Die globale Suche

Nach Abschluß seiner lokalen Suche beteiligt sich jeder Agent am globalen Vergleich. Mit dieser globalen Suche beginnen wieder die Blattknoten des Baumes, den ein Fragment bildet. Jeder Blattknoten schickt das Gewicht der kleinsten, von ihm aus dem Fragment herausführenden Kante als zusätzliche Information in der report-Nachricht an seinen Vater. Wir definieren dabei $w(x, \text{nil}) = \infty$ für alle Agenten x. Das heißt: Wenn keine Kante von x aus dem Fragment herausführt, dann schickt x den Wert ∞ an seinen Vater. Jeder andere Agent, der seine lokale Suche abgeschlossen hat, wartet auf report-Nachrichten aller Baumnachbarn außer seinem Vater. Er vergleicht die Werte in den Nachrichten mit seinem eigenen Suchergebnis und schickt den kleinsten Wert an seinen Vater weiter. Außerdem speichert er in seiner Variable b den Baumnachbarn, von dem der kleinste Wert kam, bzw. den Nachbarn außerhalb des Fragments an der kleinsten Kante, falls er selbst den kleinsten Wert hatte. Dies ist durch die modifizierte Transition t5 modelliert (siehe Abb. 7.25).



Sorten

$\mathcal{R}$: $2^{A \times (\mathbb{R} \cup \infty)}$

Aktivierungsbedingung für t5

$\vec{x} = (x, M, B, v, b) \quad \wedge \quad \mathcal{R}_{(1)} = M \setminus \{v\} \quad \wedge \quad r = \min\left(\mathcal{R}_{(2)} \cup \{w(x, b)\}\right)$
$\wedge \quad (y, r) \in \mathcal{R} \cup \{(b, w(x, b))\} \quad \wedge \quad \vec{x}' = (x, M, B, v, y)$

Alle anderen Sorten und Konstanten wie vorher.

Abb. 7.25: Die globale Suche

Am Ende der globalen Suche erhält der Namensgeber des Fragments den Wert der kleinsten aus dem Fragment herausführenden Kante. Da sich jeder Agent gemerkt hat, aus welcher Richtung der kleinste Wert kam, läßt sich nun vom Namensgeber aus der Weg zu dem Agenten zurückverfolgen, von dem die kleinste Kante ausgeht. Diesen Umstand werden wir aber erst im nächsten Verfeinerungsschritt benutzen, wenn wir das Abschicken der connect-Nachricht als verteilten Algorithmus modellieren. Wenn beim Namensgeber nur der Wert ∞ ankommt und auch von ihm keine Kante mehr aus dem Fragment herausführt, dann gibt es keine aus dem Fragment

herausführende Kante, d.h. das Fragment ist der minimale spannende Baum und der Algorithmus ist beendet.

Das Petrinetzmodell

Wir erhalten ein Petrinetzmodell von Algorithmus 11, indem wir die Transitionen t6, t7, t8 zu Σ_{10} hinzunehmen und die Anfangsmarkierung und die anderen Transitionen wie folgt verändern:

- Am Anfang ist die Menge B jedes Agenten die Menge aller seiner adjazenten Kanten und b ist der Nachbar an der kleinsten Kante.
- Bei t1 und t4 wird der Wert von b auf $\perp$ gesetzt.
- Bei t2 wird (x, y) aus B entfernt.
- Bei t3 wird die Kante (x, y) aus der Menge B des Agenten x entfernt und der Namensgeber n des Fragments merkt sich, daß die Suche abgeschlossen ist, indem er seine Variable b wieder auf den Wert $\perp$ setzt. Warum er das tut, wird erst im nächsten Verfeinerungsschritt klar.
- Transition t5 wird wie in Abb. 7.25 modifiziert.

Wir modifizieren die Addition einer Kante zu einem Agenten oder einem Fragment folgendermaßen: Für $\vec{x} = (x, M, B, v, b)$ ist

$$\vec{x} + (x, y) = (x, M \cup \{(x, y)\}, B \setminus \{(x, y)\}, v, b).$$

Außerdem definieren wir den Wegfall einer Basiskante als

$$\vec{x} - (x, y) = (x, M, B \setminus \{(x, y)\}, v, b).$$

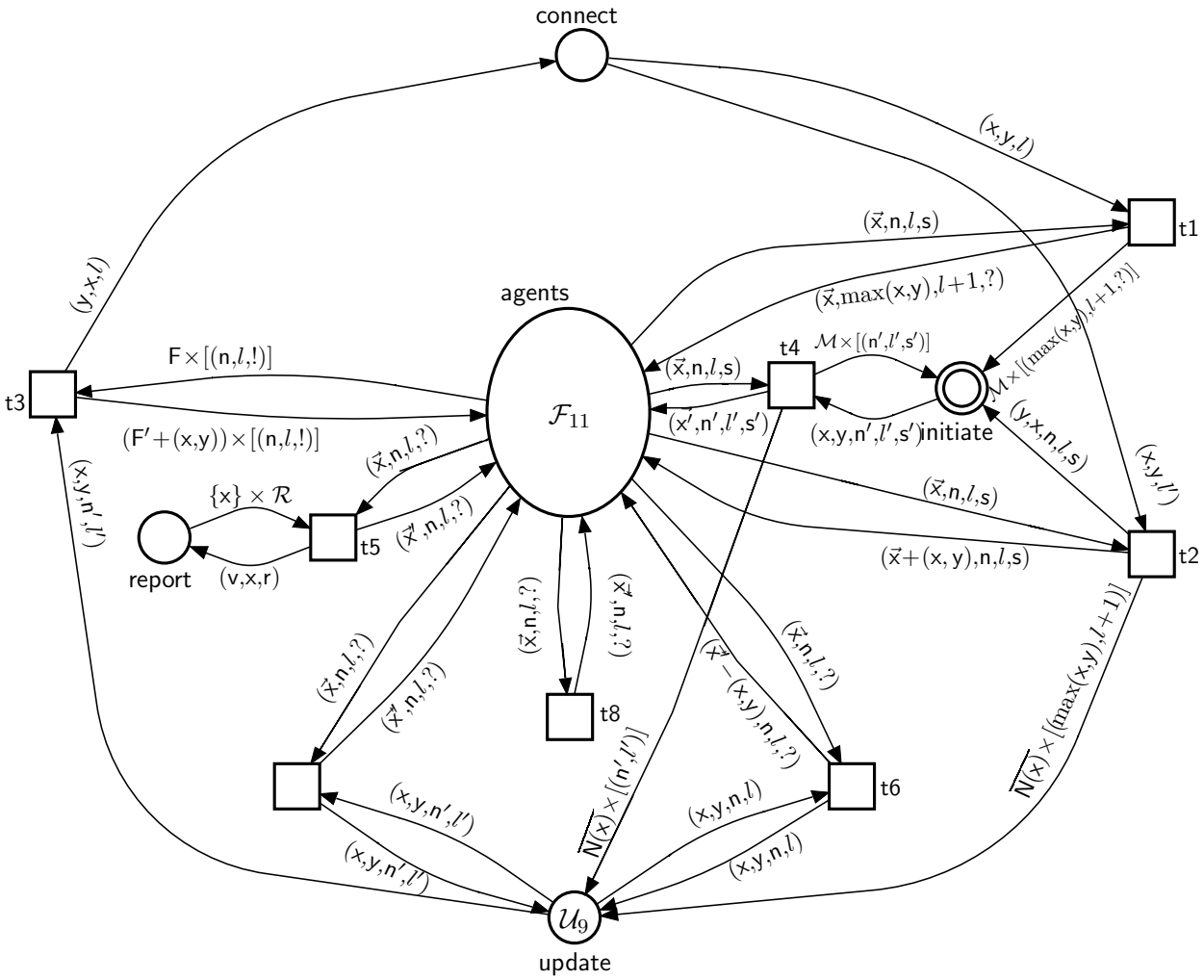
Für ein Fragment bedeutet

$$F + (x, y) = \big(F \setminus \{\vec{x}\}\big) \cup \{\vec{x} + (x, y)\}.$$

Das vollständige Petrinetzmodell von Algorithmus 11 ist in Abb. 7.26 angegeben.

Korrektheit

Σ_{11} ist eine Beobachterverfeinerung von Σ_{10} bzgl. folgender Beobachter: Obs_{10}, der Beobachter von Σ_{10} sieht die gesamte Markierung von Σ_{10}, Obs_{11}, der Beobachter von Σ_{11} sieht auch alle Stellen des Systems Σ_{11}, aber er blendet überall die neuen Daten "Basiskanten" und "beste Kante" aus, so daß er auch eine Markierung von Σ_{11} sieht. Die Transitionen t6, t7, t8 sind neue Transitionen, die nichts an den alten Variablen ändern. Alle anderen Transitionen verändern die alten Variablen genauso, wie in Σ_{10}. Wir müssen also nur zeigen, daß Σ_{11} nicht verklemmt, bevor der Algorithmus abgeschlossen ist. Eine vorzeitige Verklemmung kann nur auftreten, wenn mehrere Agenten zyklisch auf update-Nachrichten voneinander warten. Dies kann aber nicht passieren, da in diesem Zyklus die Level der Agenten von einem zum nächsten immer kleiner werden müßten.



Sorten

$\vec{\mathsf{x}}, \vec{\mathsf{x}}'$:	$\mathbb{A}\text{gent} \times 2^N \times A \times A^*$
b :	A^* mit $A^* = A \cup \{\bot, \text{nil}\}$
r :	$\mathbb{R}$
$\mathcal{R}$:	$A \times \mathbb{R}$
agents :	$\mathbb{A}\text{gent} \times 2^N \times A \times A^* \times A \times \mathbb{N} \times \mathbb{S}\text{tate}$

Konstanten

$\mathcal{F}_{11} = \{(a, \emptyset, N(a), a, \text{mo}(a)_{(2)}, a, 0, !) \mid a \in A\}$

Aktivierungsbedingungen

Es sei immer $\vec{x} = (x, M, B, v, b)$ und $\vec{x}' = (x, M', B', v', b')$
und falls nicht anders angegeben $M = M'$, $B = B'$ u.s.w.

t1 : $(x, y) \in M \ \wedge \ \mathcal{M} = \overline{M} \setminus \{(y, x)\} \ \wedge \ b' = \bot$
t3 : $l \leq l' \ \wedge \ x \in F \ \wedge \ (x, y) \notin F \ \wedge \ \forall z, z' : (z, z') \in F \to z' \in F$
$\wedge$ $(n, M, B, v, b) \in F \ \wedge \ b' = \bot$
t4 : $\mathcal{M} = \overline{M} \setminus \{(y, x)\} \ \wedge \ v' = y \ \wedge \ b' = \bot$
Alle anderen Aktivierungsbedingungen, Sorten und Konstanten wie in Abb. 7.24, Abb. 7.25 und Σ_{10}.

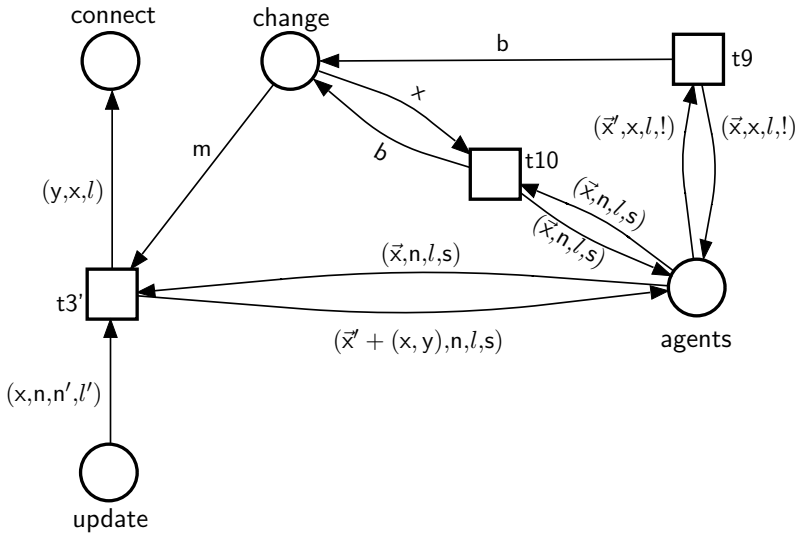
Abb. 7.26: Σ_{11}

7.12 Abschicken der connect-Nachricht

Informelle Beschreibung

Bisher waren alle Agenten eines Fragments synchron am Abschicken der connect-Nachricht beteiligt. Diese Aktion verfeinern wir nun zu einem nachrichtenbasierten Algorithmus. Dieser Teilalgorithmus beginnt nach Abschluß der globalen Suche in einem Fragment. Ausgehend vom Namensgeber des Fragments wird über die b(este Kante)-Variablen der Agenten zurückverfolgt, von welchem Agenten die kleinste Kante aus dem Fragment herausführt. Dieser Agent schickt dann die connect-Nachricht ab.

Algorithmus 12 ist ein nachrichtenbasierter Algorithmus, der verteilt den minimalen spannenden Baum berechnet. Es werden viele update-Nachrichten versendet, die nie benutzt werden.



Sorten

change : A
m : A

Aktivierungsbedingungen

Es sei immer $\vec{x} = (x, M, B, v, b)$ und $\vec{x}' = (x, M', B', v', b')$
und falls nicht anders angegeben $M = M'$, $B = B'$ u.s.w.

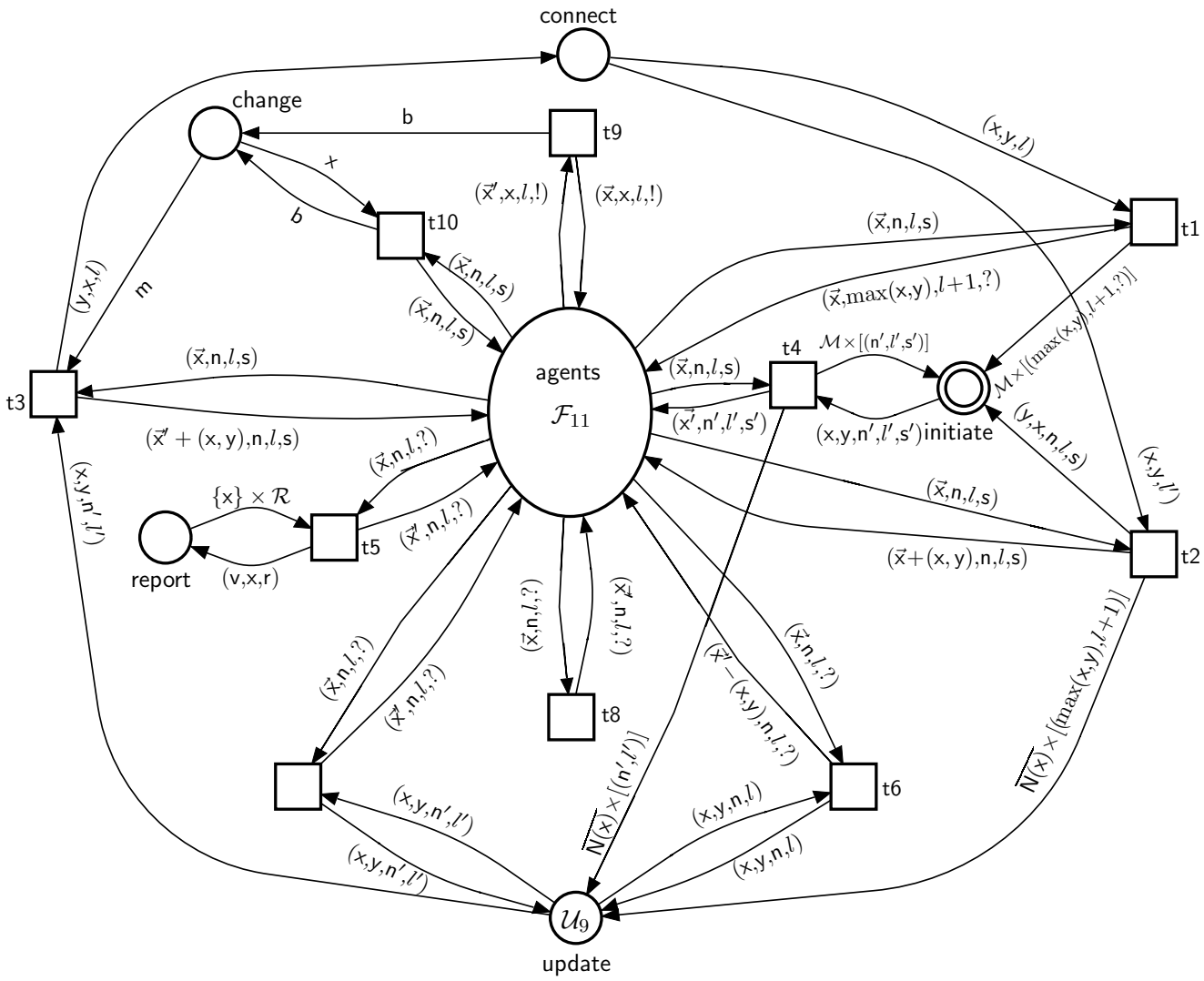
t3′ : $\big(x \neq n \ \wedge \ m = x \ \wedge \ b = y\big) \quad \vee \quad \big(x = n \ \wedge \ m = \emptyset \ \wedge \ s =! \ \wedge \ b = y \ \wedge \ b \neq \perp \ \wedge \ b' = \perp\big)$

t9 : $(x, b) \in M \ \wedge \ b' = \perp$

t10 : $(x, b) \in M$

Alle anderen Aktivierungsbedingungen, Sorten und Konstanten wie in Σ_{11}.

Abb. 7.27: Abschicken der connect-Nachricht



Alle Aktivierungsbedingungen, Sorten und Konstanten wie in Abb. 7.27 und Σ_{11}.

Abb. 7.28: Σ_{12}

Das Petrinetzmodell

Das Abschicken der connect-Nachricht wurde bisher durch Transition t3 modelliert. In Abb. 7.27 ist der Teilalgorithmus, durch den wir t3 ersetzen, als Petrinetzmodell angegeben. Der Algorithmus wird durch den Namensgeber eines Fragments ($x = n$) nach Abschluß der Suche ($s =!$ und $b \neq \perp$) gestartet. Wenn die kleinste aus dem Fragment herausführende Kante direkt vom Namensgeber ausgeht ($(x, b) \in B$), dann schickt der Namensgeber die connect-Nachricht ab und merkt sich dies, indem er b auf $\perp$ setzt (Transition t3′). In diesem Fall ist keine change-Nachricht nötig ($m = \emptyset$). Dieser Fall ist insbesondere am Anfang des Algorithmus gegeben, wenn jedes

Fragment nur aus einem einzigen Agenten besteht, der gleichzeitig der Namensgeber ist und von dem natürlich auch die kleinste Kante aus dem Fragment herausführt.

Wenn die kleinste Kante nicht von Namensgeber ausgeht ($(x, b) \in M$), dann schickt dieser eine change-Nachricht an seinen Baumnachbarn b, von dem er den besten Wert bekommen hat (Transition t9). Damit der Namensgeber diese Nachricht nicht mehrmals verschickt, setzt er beim Verschicken $b = \perp$. In einer change-Nachricht spielt der Absender keine Rolle, deshalb geben wir nur den Empfänger an.

Ein Agent, der eine change-Nachricht bekommt, überprüft, ob er den besten Wert von einem Baumnachbarn bekommen hat ($(x, b) \in M$), oder ob er selbst den besten Wert hatte ($(x, b) \in B$). Im ersten Fall schickt er die Nachricht an b weiter (Transition t10), im zweiten Fall schickt er die connect-Nachricht ab (Transition t3′).

Das vollständige Petrinetzmodell von Algorithmus 12 ist in Abb. 7.28 angegeben.

Korrektheit

Das Ersetzungsnetz aus Abb. 7.27 ist eine Transitionsverfeinerung von t3 im System Σ_{11}.

Zuerst müssen wir zeigen, daß das Netz aus Abb. 7.27 wirklich ein Ersetzungsnetz ist. Dazu setzen wir für einen beliebigen Modus β den Vorbereich der Aktion (t, β) als Anfangsmarkierung in das Netz aus Abb. 7.27 ein und zeigen daß jeder Ablauf dieses System im Zustand $(t, \beta)^-$ endet.

Danach zeigt man wieder, daß es keine "echten" Konflikte der Umgebung zu Transitionen des Ersetzungsnetzes gibt. Die einzigen Konflikte sind Beitritte, nach denen die Aktion immer noch aktiviert ist. Außerdem führt jeder Agent in jedem Ablauf des Ersetzungsnetzes höchstens eine Aktion aus.

7.13 Der GHS-Algorithmus – Tests statt Updates

Informelle Beschreibung

Wir verfeinern nun Algorithmus 12 zum GHS-Algorithmus. Im GHS-Algorithmus werden keine update-Nachrichten versendet. Statt dessen versendet ein Agent bei der lokalen Suche eine test-Nachricht über die kleinste Kante, über die er noch nichts weiß. Der Empfänger einer test-Nachricht entscheidet dann, ob er zum selben Fragment gehört wie der Absender oder nicht.

Ein Agent im Zustand find, der seine "beste Kante" noch nicht gefunden hat ($b = \perp$), schickt eine test-Nachricht mit seinem Fragmentnamen und Level über die kleinste Kante aus der Menge B der Basiskanten, über die er noch nichts weiß (Transition t11). Damit er die Nachricht nicht mehrmals verschickt, setzt er die Variable b auf den Sonderwert $\top$.

Ein Agent, der eine test-Nachricht bekommt, beantwortet diese Nachricht erst dann, wenn sein eigener Level mindestens so groß wie der Level in der Nachricht ist. Der Grund dafür ist derselbe wie früher: Ist sein Level kleiner als der in der Nachricht, dann kann es sein, daß er schon zum Fragment des Absenders gehört, aber noch nicht

die initiate-Nachricht mit dem neuen Namen und Level empfangen hat. Enthält die test-Nachricht seinen eigenen Fragmentnamen und Level, dann entfernt er die Kante aus seiner Menge B und schickt als Antwort eine reject-Nachricht zurück (Transition t12). Ist sein eigener Level mindestens genauso groß wie der in der Nachricht und ist der Fragmentname verschieden von seinem eigenen, dann schickt er eine accept-Nachricht zurück (Transition t13).

Ein Agent, der eine reject-Nachricht empfängt, entfernt die entsprechende Kante aus seiner Menge B und setzt $b = \bot$, um erneut mit der Suche zu beginnen (Transition t6). Ein Agent, der eine accept-Nachricht von einem Agenten y empfängt, setzt $b = y$ und beendet damit die lokale Suche (Transition t7).

Das Petrinetzmodell

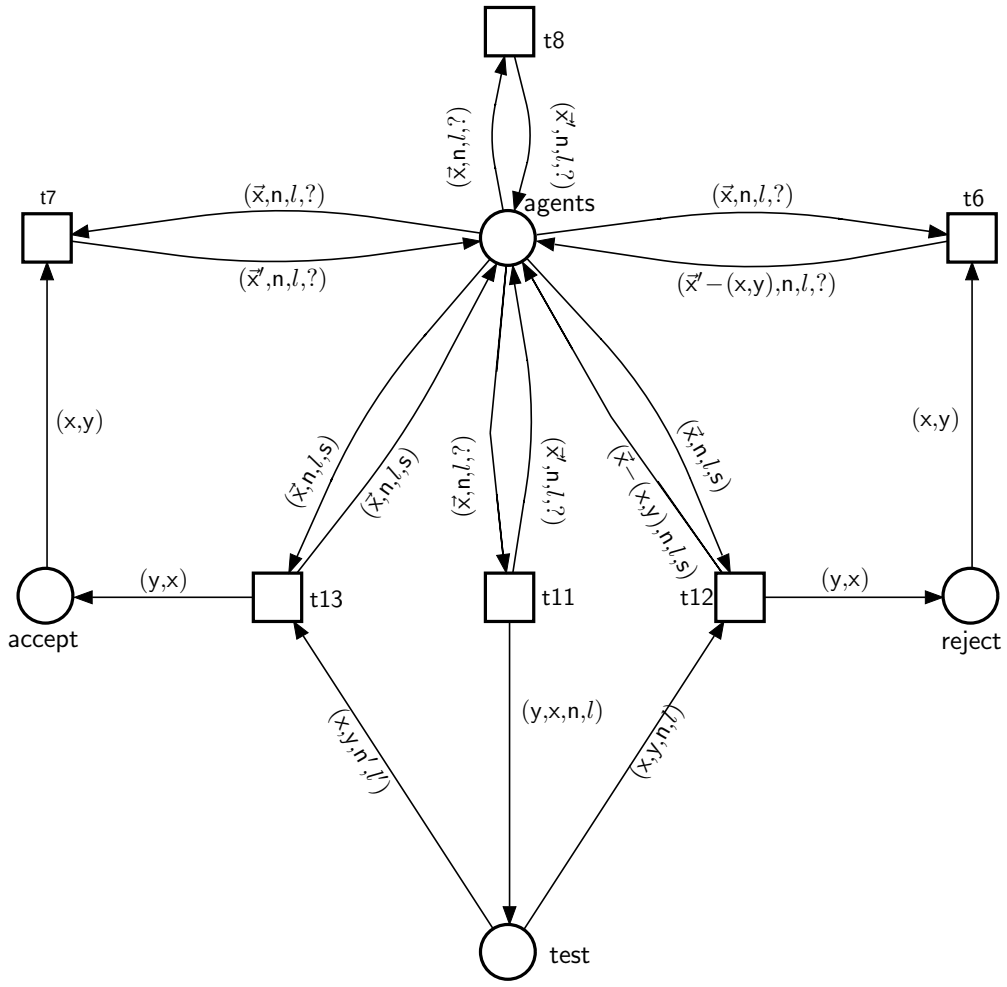
Wir erhalten ein Petrinetzmodell des GHS-Algorithmus, indem wir die Transitionen t6, t7 und die Stelle update und alle dazugehörigen Kanten aus Σ_{12} entfernen und dafür das Teilnetz aus Abb. 7.29 einfügen. Dieses Modell ist in Abb. 7.30 explizit mit allen Sorten und Aktivierungsbedingungen angegeben.

Korrektheit

Das System Σ_{13} ist eine Beobachterverfeinerung von Σ_{12} bzgl. folgender Beobachter:

- obs_{12} sieht alles außer update.
- obs_{13} sieht alles außer test, accept und reject und sieht jedes $\top$ als $\bot$.

Damit haben wir den GHS-Algorithmus vollständig hergeleitet.



Sorten

b :	$A \cup \{\bot, \top, \text{nil}\}$
test :	$N \times A \times \mathbb{N}$
accept :	N
reject :	N

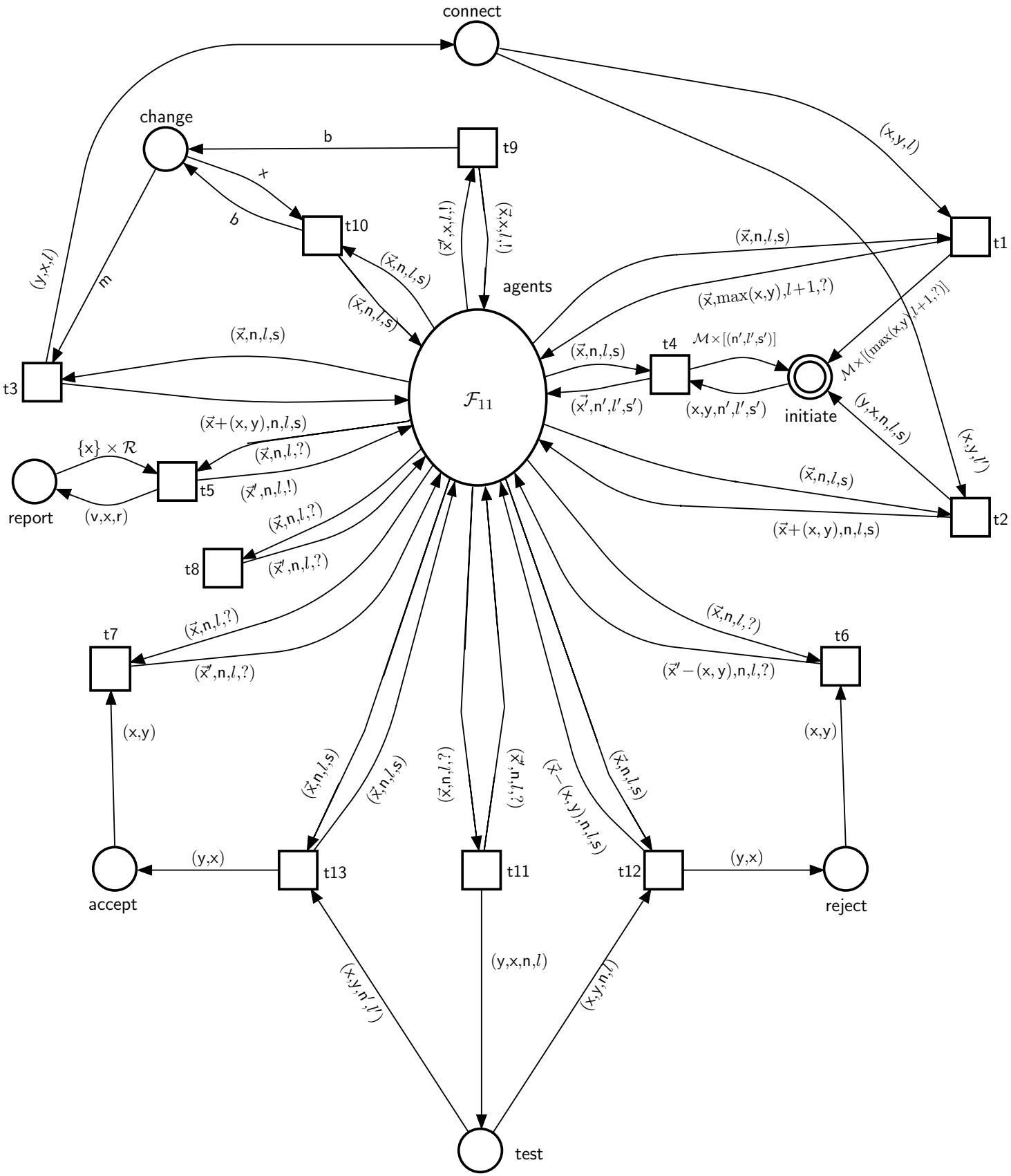
Aktivierungsbedingungen

Es sei immer $\vec{x} = (x, M, B, v, b)$ und $\vec{x}' = (x, M', B', v', b')$
und falls nicht anders angegeben $M = M'$, $B = B'$ u.s.w.

t6 :	$b' = \bot$
t7 :	$b' = y$
t11 :	$b = \bot \ \wedge \ w(x,y) = \min\{w(x,z) \mid (x,z) \in B\} \ \wedge \ b' = \top$
t12 :	true
t13 :	$l' \leq l \ \wedge \ n \neq n'$

Alle anderen Sorten und Konstanten wie vorher.

Abb. 7.29: Die lokale Suche mit Tests


connect
change
b
t9
x
b
t10
(x,y,l)
(y,x,l)
m
(x⃗,x,l,!)
(x⃗,x,l,!)
agents
(x⃗,n,l,s)
(x⃗,n,l,s)
t1
(x⃗,n,l,s)
(x⃗,max(x,y),l+1,?)
M×[(max(x,y),l+1,?)]
t4
M×[(n′,l′,s′)]
(x⃗,n,l,s)
t3
(x⃗,n,l,s)
F11
(x⃗′,n′,l′,s′)
(x,y,n′,l′,s′)
initiate
(y,x,n,l,s)
(x,y,l)
(x⃗+(x,y),n,l,s)
{x} × R
(x⃗,n,l,?)
t5
(x⃗′,n,l,!)
report
(v,x,r)
(x⃗,n,l,s)
t2
(x⃗+(x,y),n,l,s)
(x⃗,n,l,?)
t8
(x⃗′,n,l,?)
(x⃗,n,l,?)
t7
(x⃗′,n,l,?)
(x⃗,n,l,?)
t6
(x⃗′−(x,y),n,l,?)
(x,y)
(x,y)
(x⃗,n,l,s)
(x⃗,n,l,s)
(x⃗,n,l,?)
(x⃗′,n,l,?)
(x⃗−(x,y),n,l,s)
(x⃗,n,l,s)
(y,x)
t13
t11
t12
(y,x)
accept
reject
(y,x,n,l)
(x,y,n′,l′)
(x,y,n,l)
test

Kommunikationsnetzwerk: (A, N, w)

Sorten

$\mathbb{A}\text{gent}$:	$A \times 2^N$
$\mathsf{x}, \mathsf{y}, \mathsf{v}, \mathsf{n}$:	A
$\vec{\mathsf{x}}, \vec{\mathsf{x}}'$:	$\mathbb{A}\text{gent} \times 2^N \times A \times A^*$
M, B :	2^N
b :	A^* mit $A^* = A \cup \{\bot, \top, \text{nil}\}$
l :	$\mathbb{N}$
s :	$\mathbb{S}\text{tate}$
r :	$\mathbb{R}$
$\mathcal{M}$:	2^N
$\mathcal{R}$:	$2^{A \times (\mathbb{R} \cup \infty)}$
m :	A
agents :	$\mathbb{A}\text{gent} \times 2^N \times A \times A^* \times A \times \mathbb{N} \times \mathbb{S}\text{tate}$
$\mathsf{connect}$:	$N \times \mathbb{N}$
$\mathsf{initiate}$:	$N \times A \times \mathbb{N} \times \mathbb{S}\text{tate}$
test :	$N \times A \times \mathbb{N}$
accept :	N
reject :	N
report :	$N \times \mathbb{R}$
change :	A

Konstanten

$\mathcal{F}_{11} = \{(a, \emptyset, N(a), a, \text{mo}(a)_{(2)}, a, 0, !) \mid a \in A\}$
$? = \mathsf{find}$
$! = \mathsf{found}$

Aktivierungsbedingungen

Es sei immer $\vec{x} = (x, M, B, v, b)$ und $\vec{x}' = (x, M', B', v', b')$
und falls nicht anders angegeben $M = M'$, $B = B'$ u.s.w.

$\mathsf{t1}$:	$(x, y) \in M \ \wedge \ \mathcal{M} = \overline{M} \setminus \{(y, x)\} \ \wedge \ b' = \bot$
$\mathsf{t2}$:	$l' < l$
$\mathsf{t3}$:	$\big(x \neq n \ \wedge \ m = x \ \wedge \ b = y\big) \quad \vee \quad \big(x = n \ \wedge \ m = \emptyset \ \wedge \ s = ! \ \wedge \ b = y \ \wedge \ b \neq \bot \ \wedge \ b' = \bot\big)$
$\mathsf{t4}$:	$\mathcal{M} = \overline{M} \setminus \{(y, x)\} \ \wedge \ v' = y \ \wedge \ b' = \bot$
$\mathsf{t5}$:	$\mathcal{R}_{(1)} = M \setminus \{v\} \ \wedge \ b \neq \bot \ \wedge \ r = \min\big(\mathcal{R}_{(2)} \cup \{w(x, b)\}\big) \ \wedge \ (y, r) \in \mathcal{R} \cup \{(b, w(x, b)\}) \ \wedge \ b' = y$
$\mathsf{t6}$:	$b' = \bot$
$\mathsf{t7}$:	$b' = y$
$\mathsf{t8}$:	$b = \bot \ \wedge \ B = \emptyset \ \wedge \ b' = \text{nil}$
$\mathsf{t9}$:	$(x, b) \in M \ \wedge \ b' = \bot$
$\mathsf{t10}$:	$(x, b) \in M$
$\mathsf{t11}$:	$b = \bot \ \wedge \ w(x, y) = \min\{w(x, z) \mid (x, z) \in B\} \ \wedge \ b' = \top$
$\mathsf{t12}$:	true
$\mathsf{t13}$:	$l' \leq l \ \wedge \ n \neq n'$

Abb. 7.30: Ein Petrinetzmodell des GHS-Algorithmus

8 Diskussion der Herleitung

8.1 Die anderen GHS-Beweise

Der Beweis von Chou und Gafni

Chou und Gafni schlagen in [CG88] eine neue Beweistechnik vor: *stratified decomposition*. Sie betrachten verteilte Algorithmen, die abstrakt gesehen in nacheinander folgenden Phasen ablaufen. Diese Phasen laufen im konkreten verteilten Algorithmus teilweise nebenläufig zueinander ab. Sie schlagen vor, die Aktionen eines solchen Algorithmus in verschiedene Schichten (*layer*) aufzuteilen. Sie definieren dann, wann ein solcher in Schichten aufgeteilte Algorithmus *serialisierbar* ist. Auf unser formales Modell übertragen ist das genau dann der Fall, wenn man in jedem Ablauf des Algorithmus Schnitte finden kann, so daß zwischen diesen Schnitten genau die Aktionen einer Schicht liegen [Peu99]. Das Ergebnis ist intuitiv ausgedrückt folgendes: Wenn ein Algorithmus in Schichten serialisierbar ist, dann kommt der Algorithmus zu dem gleichen Ergebnis, wie der Algorithmus, in dem dieselben Schichten streng hintereinander ausgeführt werden. Die Arbeit von Chou und Gafni ist eine Weiterentwicklung der Idee der *Communication Closed Layers* von Elrad und Francez [EF82].

Chou und Gafni beweisen dann eine einfachere Version der GHS-Algorithmus, in der genau determiniert ist, wann welches Fragment einem Fragment mit höherem Level beitritt. Durch diesen Ausschluß von Nichtdeterminismus ist es möglich, den Algorithmus in Schichten zu unterteilen, so daß jedem möglichen Fragmentlevel eine Schicht zugeordnet wird. Die Aktionen der Schicht l sind dann genau die möglichen Aktionen eines Fragments mit dem Level l. Der originale GHS-Algorithmus ist nicht in diesen Schichten serialisierbar, aber für die einfachere Version in [CG88] funktioniert die Beweismethode *stratified decomposition*.

Der Beweis von Stomp und de Roever

Auch Stomp und de Roever [SdR87, SdR94] demonstrieren am GHS-Algorithmus eine neue Beweismethode: *sequentially phased reasoning*. Auch diese Methode entwickelt die Idee der *communication closed layers* weiter. Ähnlich wie *stratified decomposition* ist diese Beweismethode auf Algorithmen anwendbar, die abstrakt gesehen in aufeinander folgenden Phasen ablaufen, deren Phasen aber im konkreten Algorithmus nebenläufig zueinander ausgeführt werden.

Sequentially phased reasoning ist flexibler als *stratified decomposition*, denn die Teilaufgaben (Phasen) beziehen sich nicht wie die Schichten global auf den ganzen Algorithmus, sondern nur auf Gruppen von Agenten. Es ist nur gefordert, daß Agenten,

die an verschiedenen Teilaufgaben des Algorithmus beteiligt sind, diese Teilaufgaben in der gleichen Reihenfolge durchlaufen.

Der vollständige Beweis des GHS-Algorithmus ist in der Dissertation [Sto89] angegeben. Der GHS-Algorithmus wird in einzelne Teilaufgaben zerlegt, deren Korrektheit isoliert von den anderen Teilaufgaben bewiesen wird. Es wird gezeigt, daß diese Teilaufgaben im Zusammenspiel das Prinzip des *sequentially phased reasoning* erfüllen. Dieser aus Teilaufgaben zusammengefügte Algorithmus ist ebenfalls eine Version des GHS-Algorithmus, in dem der Nichtdeterminismus des Beitritts eines Fragments zu einem Fragment mit höherem Level ausgeschlossen ist. In einem weiteren Schritt wird dieser Nichtdeterminismus eingeführt.

Der Beweis von Janssen und Zwiers

Janssen und Zwiers [JZ92] wollen das gleiche Problem lösen, wie Chou, Gafni, Stomp und de Roever: Algorithmen verstehen, die in nebenläufigen Phasen ablaufen, die abstrakt gesehen hintereinander ausgeführt werden. Sie wählen dazu jedoch einen algebraischen Ansatz. Sie definieren in [JZ92] einen neuen Programmoperator $\bullet$. Dieser Operator heißt *layer composition* und ist eine Mischung aus sequentieller und paralleler Programmkomposition.

Für zwei Programme P und Q bedeutet $P \bullet Q$, daß unabhängige Aktionen von P und Q nebenläufig zueinander ausgeführt werden. Falls es allerdings einen Konflikt zwischen Aktionen von P und Q gibt, wird die Aktion von P ausgeführt.

Der vollständige Beweis des GHS-Algorithmus mit Hilfe von *layer composition* ist in der Dissertation [Jan94] angegeben. In mehreren Verfeinerungsschritten wird eine um Nichtdeterminismus reduzierte Version des GHS-Algorithmus hergeleitet. Janssen erklärt, daß es vom Algorithmus, den er bewiesen hat, zum GHS-Algorithmus noch ein weiterer großer Schritt ist, in dem man den im GHS-Algorithmus vorkommenden Nichtdeterminismus bei der Annahme der connect-Nachrichten einführen müßte. Janssen stellt fest, daß die Einführung von Nichtdeterminismus im Verfeinerungsprozeß ungewöhnlich ist. Er kann in seinem formalen Modell nicht damit umgehen und muß deshalb diesen letzten Verfeinerungsschritt weglassen.

Der Beweis von Welch, Lamport und Lynch

Im Gegensatz zu den Arbeiten [SdR87, JZ92, CG88], die alle auf dem Konzept der *Communication Closed Layers* von Elrad und Francez [EF82] beruhen, ist in [WLL88] ein ganz anderer Ansatz zum Beweis des GHS-Algorithmus dargestellt.

Dort wird ein Verband (lattice) aus verschiedenen Spezifikationen angegeben mit einer einfachen Anfangsspezifikation und der Endspezifikation GHS-Algorithmus. Dann wird gezeigt, daß jede Spezifikation in diesem Verband ihre Vorgängerspezifikation bzgl. der Sicherheitseigenschaft simuliert. Außerdem wird gezeigt, daß es einen Pfad durch den Verband gibt, so daß jede Spezifikation im Pfad ihre Vorgängerspezifikation bzgl. der Lebendigkeitseigenschaft simuliert. Daraus kann man dann folgern, daß der GHS-Algorithmus korrekt ist. In [WLL88] ist diese Idee als *extended abstract* angegeben. Diese Idee wurde auch hier in einer Dissertation [Wel88] vollständig ausgeführt. Der Beweis ist technisch sehr aufwendig und über 200 Seiten lang.

Auch hier wurde am GHS-Algorithmus eine neue Beweismethode ausprobiert. Lamport gab im persönlichen Gespräch zu, daß die Idee zwar gut aussah, aber tatsächlich viele zusätzliche Beweisobligationen erzeugt wurden, durch die der Beweis sehr aufwendig wurde.

Der Beweis von Hesselink

In [Hes99b, Hes99a] ist der Beweis des GHS-Algorithmus mit Hilfe eines Theorembeweisers beschrieben. Er benutzt den Theorembeweiser Nqthm von Boyer and Moore [BM88]. Der Beweis beruht auf Invarianten und einer Abstiegsfunktion (*decreasing variant function*). Für den Beweis wurden 166 Invarianten benutzt und 32 000 Programmzeilen eingegeben.

Unser Beweis

Der in dieser Arbeit angegebene Beweis ist durch die Beweise in [SdR87, JZ92, CG88] inspiriert. Die Verfeinerungshierarchie unterscheidet sich jedoch von diesen Beweisen in zwei wesentlichen Punkten: Erstens ist der Nichtdeterminismus bei einem Beitritt eines Fragments zu einem Fragment mit größerem Level in allen Algorithmen der Verfeinerungshierarchie bereits enthalten. Transitionsverfeinerung ist eine Methode, mit der man diese Art von Nichtdeterminismus behandeln kann, denn es ist nicht deterministisch, wie oft und in welcher kausalen Ordnung mit anderen Transitionen eine (verfeinerte) Transition in einem Ablauf auftritt. Der zweite wesentliche Unterschied ist die Einführung von update-Nachrichten als Hilfsvariablen in der Verfeinerungshierarchie.

8.2 Unterschiede zum originalen Algorithmus

Der originale Algorithmus in [GHS83] ist in einer Pseudocode-Notation angegeben. Abgesehen vom formalen Modell gibt es noch einige andere Unterschiede.

Unterschied 1: FIFO-Kanäle

Im originalen Algorithmus gibt es zwischen zwei benachbarten Agenten je einen FIFO-Kanal. Alle Nachrichten eines Agenten an einen Nachbarn, werden von Nachbarn in derselben Reihenfolge empfangen, in der sie abgeschickt wurden. Unser Modell ist allgemeiner und zeigt, daß diese Einschränkung nicht nötig ist. Wir haben nur für die initiate-Nachrichten verlangt, daß beim Empfangen die Reihenfolge erhalten bleibt.

Wir können diese FIFO-Stelle sogar durch eine normale Stelle ersetzen, wenn wir dafür in der Aktivierungsbedingung von t4 fordern, daß ein Agent nur Nachrichten mit einem höheren Level als seinem eigenen empfängt. Die veralteten Nachrichten bleiben dann einfach liegen. Dadurch brechen alte Nachrichtenwellen durch das Fragment in dem Moment ab, wenn sie von einer neueren Nachrichtenwelle überholt werden. Leider ist es uns nicht gelungen, einen intuitiven Beweis für den Algorithmus völlig ohne FIFO-Stellen zu finden.

Unterschied 2: Schlingenfreies Kommunikationsnetzwerk

In unserer Definition eines Kommunikationsnetzwerks (siehe Seite 56), haben wir ausgeschlossen, daß ein Agent eine Kante zu sich selbst hat. In [GHS83] ist dies nicht ausgeschlossen, aber Stomp hat in [Sto89] gezeigt, daß der Algorithmus dann nicht korrekt ist. Wenn ein Agent eine Kante zu sich selbst hat und diese auch noch das kleinste Gewicht von allen ihm adjazenten Kanten hat, dann berechnet der Agent die Kante gleich zu Beginn des Algorithmus als Kante des minimalen spannenden Baumes. Diese Kante kann aber nicht zum minmalen spannenden Baum gehören, denn wenn man diese Kante aus dem Baum entfernt, erhält man immer noch einen spannenden Baum, aber mit kleinerem Gesamtgewicht.

Unterschied 3: Vereinigung

Im originalen Algorithmus tauschen die beiden Agenten an der Vereinigungskante bei einer Vereinigung noch gegenseitig initiate-Nachrichten aus, bevor die Nachricht im Fragment weiterverbreitet wird. In unserem Modell sparen wir zwei Nachrichten pro Vereinigung, da jeder Agent an der Vereinigungskante direkt die Nachricht in seinem Teil des Fragments weiterverbreitet, ohne noch eine Nachricht an den Agenten auf der anderen Seite der Vereinigungskante zu schicken.

Diese geringfügige Optimierung des originalen Algorithmus wurde auch in [SdR94, Tel94, Lyn96] angewendet.

Unterschied 4: Der Identifikator eines Fragments

Im originalen Algorithmus ist der Name des Fragments gleich dem Gewicht der Vereinigungskante, über die das Fragment entstanden ist. Da die Kantengewichte eindeutig sind, ist der auf diese Art gebildete Name ein eindeutiger Identifikator.

In unserem Modell haben wir in jedem Fragment einen ausgezeichneten Knoten, den Namensgeber des Fragments. Der Name dieses Agenten ist auch der Fragmentname. Damit dieser Name ein eindeutiger Identifikator ist, benutzen wir die zusätzliche Annahme, daß es eine partielle Ordnung auf den Namen der Agenten gibt, so daß die Namen von je zwei Nachbarn geordnet sind (vgl. Fußnote auf Seite 115). Diese Annahme ist keine große Einschränkung und trägt zur intuitiveren Darstellung des Algorithmus bei. Diese Veränderung des originalen Algorithmus wurde auch schon in [Lyn96, Tel94] bei der Darstellung des Algorithmus benutzt.

Abschließende Bemerkungen

In dieser Arbeit haben wir mit dem Begriff der Transitionsverfeinerung einen neuen Verfeinerungsbegriff vorgestellt, der auf der kausalen Ordnung von Aktionen in einem Ablauf beruht. Wir haben diskutiert, wie man beweist, daß die Ersetzung einer einzelnen Aktion durch ein komplexeres Teilsystem eine Transitionsverfeinerung ist.

Wie andere Verfeinerungskonzepte verkürzt oder vereinfacht Transitionsverfeinerung einen Beweis nicht notwendigerweise. Verfeinerungskonzepte sind Strukturierungshilfen, durch die ein Beweis nachvollziehbar wird und die zum Verständnis eines Algorithmus beitragen. Wir haben anhand des GHS-Algorithmus gezeigt, daß zur Herleitung komplexer verteilter Algorithmen eine Kombination verschiedener Verfeinerungsbegriffe sinnvoll ist.

Das in dieser Arbeit hergeleitete Modell des GHS-Algorithmus wurde am Lehrstuhl von Prof. Reisig mit Hilfe des Petrinetz-Kerns [KW99] implementiert und für verschiedene feste Kommunikationsnetzwerke getestet.

In [BKV99] wurde gezeigt, daß die in der Logik von DAWN ausgeführten Beweise automatisch überprüft werden können. Am Lehrstuhl von Prof. Reisig wurde ein Werkzeug entwickelt, um solche Beweise automatisch zu überprüfen. Es ist möglich, dieses Werkzeug um Verfeinerungskriterien zu erweitern und die kleineren Beweisziele maschinell überprüfen lassen, um einen langen, aber gut strukturierten Beweis überzeugender darzustellen.

Zum Abschluß diskutieren wir nun, wie der in dieser Arbeit vorgestellte Ansatz zur Verifikation verteilter Algorithmen mit anderen Arbeiten auf diesem Gebiet zusammenhängt und wie er in zukünftigen Arbeiten weitergeführt werden könnte und weitergeführt wird.

Transitionsverfeinerung und Communication Closed Layers

Mehr als von anderen Verfeinerungsbegriffen wurde der Begriff Transitionsverfeinerung durch die von Elrad und Francez [EF82] eingeführten *Communication Closed Layers* und darauf aufbauenden Arbeiten [Zwi98, SdR94, JMZ91, CG88] inspiriert. In diesen Arbeiten wird ein Algorithmus in verschiedene Phasen zerlegt, die von einem abstrakten logischen Standpunkt aus hintereinander ausgeführt werden, aber im konkreten Algorithmus teilweise nebenläufig zueinander ablaufen. In [Peu99] wird gezeigt, daß man dieses Prinzip adäquat mit verteilten Abläufen beschreiben kann. Ein Algorithmus läuft genau dann in Phasen ab, wenn die Phasen in jedem Ablauf durch Schnitte voneinander getrennt werden können.

Transitionsverfeinerung kann in gewisser Weise als Verallgemeinerung dieses Prinzips angesehen werden. Wir können jede Transition eines Systems als eine Phase anse-

hen. Wenn wir dann mehrere Transitionen simultan ersetzen, werden diese teilweise nebenläufig zueinander ausgeführt. Zu jedem verteilten Ablauf gibt es eine Sequentialisierung, in der diese Phasen hintereinander ausgeführt werden. Die Reihenfolge kann jedoch durch Nichtdeterminismus im System in jedem Ablauf anders sein. Insbesondere ist nicht determiniert, wie oft eine Phase in einem Ablauf stattfindet oder ob sie überhaupt stattfindet.

Kompositionale Aspekte

Wir haben in Abschnitt 5.2 gezeigt, daß bei einer Transitionsverfeinerung temporale Eigenschaften des Ausgangssystems erhalten bleiben. Es gilt aber noch mehr. Bei einer Transitionsverfeinerung werden Eigenschaften des Ersetzungsnetzes und Eigenschaften des Ausgangssystems in nicht-trivialer Weise miteinander kombiniert.

Wir betrachten noch einmal das Beispiel von Seite 78. Die Invariante $\Box\,\mathsf{A} + \mathsf{B} = 1$ des Ausgangssystem wird zwar durch das Ersetzungsnetz zerstört, aber im verfeinerten System gilt die neue Invariante $\Box\,\mathsf{A} + \mathsf{B} + \mathsf{C} = 1$. Diese Invariante setzt sich aus der Invariante des Ausgangssystems und der Invariante $\Box\,\mathsf{A} + \mathsf{B} + \mathsf{C} = 1$ des Ersetzungsnetzes mit der Anfangsmarkierung t^- zusammen.

Auch für Lebendigkeitseigenschaften kann man kompositionale Muster bei Transitionsverfeinerung erkennen. Im Ausgangssystem in unserem Beispiel gilt $\Diamond\,\mathsf{B}$ und im verfeinerten System gilt zusätzlich $\Diamond\,\mathsf{C}$. Oft ist es so, daß die Lebendigkeitseigenschaften des Ersetzungsnetzes auch im verfeinerten System gelten. Dies gilt aber zum Beispiel nicht mehr, wenn die Transition, die verfeinert wird, eine tote Transition des Ausgangssystem ist. Die Transition kann im Ausgangssystem nie schalten, deshalb schaltet auch keine Transition des Ersetzungsnetzes und die Lebendigkeitseigenschaft des Ersetzungsnetzes gilt im verfeinerten System nicht.

Die genaue Analyse dieser kompositionalen Aspekte von Transitionsverfeinerung ist ein weiteres Forschungsgebiet der Autorin. Dabei wird sie auf der von Kindler in [Kin97] vorgeschlagenen kompositionalen Semantik für Petrinetze aufbauen. Man kann das Ersetzungsnetz und die Umgebung als komponierbare Komponenten in dieser Semantik ansehen.

Syntaktische Beweiskriterien

Durch die Sätze 2.2 und 2.3 haben wir bereits ein starkes, aber immer noch semantisches Beweiskriterium für Transitionsverfeinerung. Dieses Beweiskriterium ist insbesondere bei Anwendung auf algebraische Transitionsverfeinerung noch nicht vollständig zufriedenstellend. Ein weiteres interessantes Forschungsgebiet ist deshalb die Suche nach syntaktischen Beweiskriterien, so daß z.B. auch der Beweis der Transitionsverfeinerungsschritte im GHS-Algorithmus überzeugender darstellbar ist.

Ersetzungsnetze mit Anfangsmarkierung

In unserer Definition eines Ersetzungsnetzes haben wir nicht zugelassen, daß neue Stellen (also Stellen die nicht zum Vor- oder Nachbereich von t gehören) initial markiert sind. Manchmal ist es jedoch sinnvoll, zu erlauben, daß ein Ersetzungsnetz

von Anfang an markiert ist. In diesem Abschnitt erläutern wir die Probleme, die dabei entstehen können anhand des Produzent-Konsument-Beispiels aus [Rei98].

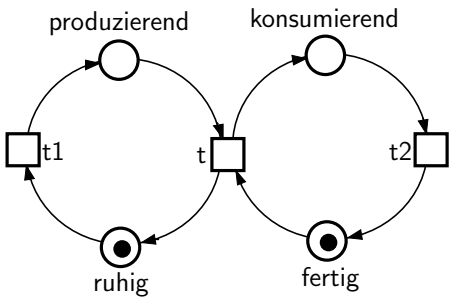


Abb. 8.1: Das System Σ: Produzent und Konsument – stark synchronisiert.

Das System besteht aus zwei Agenten, einem Produzenten, der entweder ruhig ist oder produziert und einem Konsumenten, der entweder konsumiert oder fertig ist (siehe Abb. 8.1). Die beiden Agenten haben eine synchrone Transition: Sie können nur gemeinsam vom Zustand produzierend und fertig in den Zustand ruhig und konsumierend übergehen. Wir ersetzen diese Transition durch einen Puffer, der eine Marke speichern kann (Abb. 8.2).

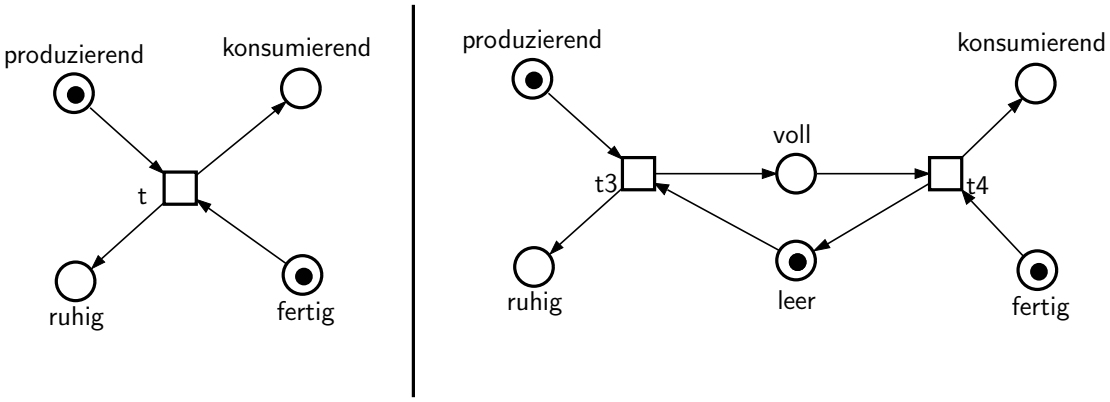


Abb. 8.2: Transition t und der Puffer als Ersetzungsnetz

Anfangs ist der Puffer leer. Wenn $t2$ schaltet, wird der Puffer voll und dann durch Schalten von $t3$ wieder leer. Wir könnten nun den Begriff Transitionsverfeinerung dahingehend erweitern, daß wir initial markierte Ersetzungsnetze zulassen, wenn die neuen Stellen des Ersetzungsnetzes nach einem Ablauf des Ersetzungsnetzes genau so markiert sind wie vorher. In unserem Beispiel ist das der Fall.

Wenn wir die Transition t durch den Puffer ersetzen, erhalten wir das System in Abb. 8.3. Wir können die Definition der Expansion eine verteilten Ablaufs (und damit auch die Definition der Transitionsverfeinerung) leicht so erweitern, daß der Puffer eine Transitionsverfeinerung von t im System Σ ist. In der erweiterten Definition der Expansion eines Ablaufs müssen wir dann nicht nur darauf achten, daß die Abläufe des Ersetzungsnetzes ordentlich mit dem Ablauf des Ausgangssystems verbunden werden, sondern auch, daß verschiedene Abläufe des Ersetzungsnetzes

ordentlich miteinander verbunden werden. In unserem Beispiel liegt nach dem ersten Ablauf des Ersetzungsnetzes eine Marke auf der Stelle leer. Diese Marke wird im zweiten Ablauf des Ersetzungsnetzes verbraucht.

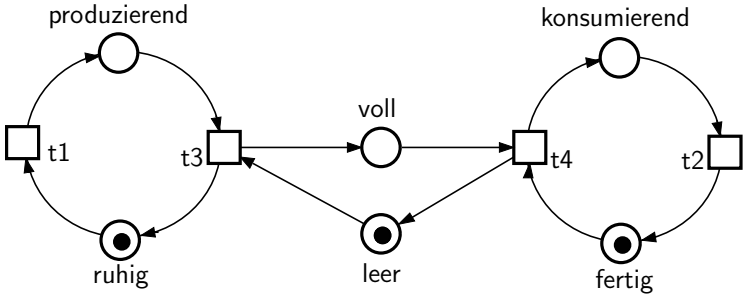


Abb. 8.3: Σ': Produzent und Konsument durch einen Puffer synchronisiert

Ein Problem entsteht, wenn wir zwei ununterscheidbare Produzenten und zwei ununterscheidbare Konsumenten habe, d.h. je zwei Marken auf den Stellen ruhig und fertig. Im stark synchronisierten Produzenten-Konsumenten-System in Abb. 8.3 könnte dann die Transition t nebenläufig zu sich selbst schalten, d.h. nebenläufig zueinander synchronisieren sich je ein Produzent und ein Konsument.

Wenn wir auch in diesem System die Transition t durch einen Puffer mit der Kapazität 1 ersetzen, dann ist diese nebenläufige Synchronisation nicht möglich. Die Aktionen der beiden Produzenten werden durch den Puffer sequentialisiert. In diesem Fall ist der Puffer nach der erweiterten Definition eine Transitionsverfeinerung, aber die Eigenschaften des Ausgangssystems bleiben offensichtlich nicht erhalten. Die Eigenschaften, die verloren gehen, sind allerdings reine Halbordnungseigenschaften und alle temporalen Eigenschaften, die in den sequentiellen Abläufen des Ausgangssystems gelten, bleiben erhalten.

Wir haben nun mehrere Möglichkeiten: Wir können Transitionsverfeinerung mit Anfangsmarkierungen im Ersetzungsnetz so definieren, wie hier angedeutet und dann zeigen, daß diese Transitionsverfeinerung weniger Eigenschaften erhält als Transitionsverfeinerung ohne Anfangsmarkierung, oder wir können uns in der Definition von Transitionsverfeinerung mit Anfangsmarkierung gleich auf sichere Systeme beschränken.

Die Untersuchung dieser verschiedenen Möglichkeiten ist ein weiteres Forschungsvorhaben der Autorin.

A Anhang

A.1 Beweis von Satz 2.3

Sei $\rho' = ((B', E', \lessdot'), r')$ ein Ablauf von $\Sigma\big[t \to N_E\big]$ und sei $\wp$ eine Partition aller N_E-Ereignisse von ρ', die die Bedingungen (i)(a) und (ii) aus Satz 2.2, aber nicht Bedingung (i)(b) erfüllt.

Bevor wir den Beweis führen, legen wir noch folgende Notationen fest:

- Sei A eine Äquivalenzklasse der Partition und sei C ein Schnitt in ρ'. Der Schnitt C teilt A, wenn es $a_1, a_2 \in A$ gibt, so daß $a_1 < C$ und $a_2 > C$. Wir bezeichnen dies mit $C \mid A$. Der Schnitt C teilt A nicht, wenn entweder $A < C$ oder $A > C$ gilt. Wir bezeichnen dies mit $C \nmid A$.
- Für zwei Schnitte C_1 und C_2 definieren wir das Supremum der beiden Schnitte als Menge aller Elemente aus $C_1 \cup C_2$, die keinen Nachfolger in $C_1 \cup C_2$ haben, d.h.

  $$\sup(C_1, C_2) = \{c \in C_1 \cup C_2 \mid \forall c' \in C_1 \cup C_2 : c' \leq c \vee c' \operatorname{co} c\}.$$

  Analog definieren wir das Infimum durch

  $$\inf(C_1, C_2) = \{c \in C_1 \cup C_2 \mid \forall c' \in C_1 \cup C_2 : c \leq c' \vee c \operatorname{co} c'\}.$$

- Für zwei Mengen A und B von Ereignissen aus ρ' schreiben wir $A \operatorname{co} B$, wenn $a \operatorname{co} b$ für alle $a \in A$ und $b \in B$ gilt.

Bemerkung A.1 Das Supremum und das Infimum von zwei Schnitten sind wieder Schnitte und es gilt: $\inf(C_1, C_2) \leq \sup(C_1, C_2)$. ⋆

Seien A und B zwei beliebige verschiedene Äquivalenzklassen aus $\wp$ und seien C_1, C_2 zwei Schnitte, die die Bedingungen (i)(a) und (ii) für A erfüllen und D_1, D_2 die dementsprechend zu B gehörenden Schnitte. Zuerst stellen wir fest:

Behauptung 1: Für jede Äquivalenzklasse G von $\wp$ gilt:

$$C_1 \mid G \iff C_2 \mid G \qquad \text{(und damit auch } D_1 \mid G \iff D_2 \mid G)$$

Beweis: (Für C_1, C_2. Der Beweis für D_1, D_2 ist analog, denn A und B sind beide beliebig gewählt.) Wenn nur einer der beiden Schnitte G teilt und der andere nicht, dann gibt es ein Element $g \in G$ das zwischen den beiden Schnitten liegt. Dies ist jedoch ein Widerspruch dazu, daß zwischen den beiden Schnitten genau die Elemente von A liegen ((i)(a)).

Behauptung 2: Wir können die Ereignisse aus $A \cup B$ so zu zwei neuen Äquivalenzklassen $\tilde{A}$ und $\tilde{B}$ umordnen, daß

(i) $A \cup B = \tilde{A} \cup \tilde{B}$

(ii) Für $\tilde{A}$ und $\tilde{B}$ ist Bedingung (ii) aus Satz 2.2 erfüllt

(iii) Es gibt Schnitte $\tilde{C_1}, \tilde{C_2}$ zu $\tilde{A}$ und Schnitte $\tilde{D_1}, \tilde{D_2}$ zu $\tilde{B}$, so daß (i)(a) aus Satz 2.2 erfüllt ist und $C_1 \nmid B$ und $D_1 \nmid A$

(iv) für alle Äquivalenzklassen $G \neq A, B$ gilt:

$$C_1 \mid G \quad \Longleftrightarrow \quad \tilde{C_1} \mid G \text{ und}$$
$$D_1 \mid G \quad \Longleftrightarrow \quad \tilde{D_1} \mid G$$

Um diese Behauptung zu beweisen, müssen wir verschiedene Fälle betrachten. Der einfachste Fall ist natürlich, daß $C_1 \nmid B$ und $D_1 \nmid A$. In diesem Fall müssen wir die Ereignisse gar nicht umordnen. Wir wählen $A = \tilde{A}$, $C_1 = \tilde{C_1}$ u.s.w.

Der Fall, daß $C_1 \nmid B$ (also auch $C_2 \nmid B$), aber $D_1 \mid A$ (also auch $D_2 \mid A$) gilt, kann in sicheren Systemen nicht auftreten. Angenommen, dieser Fall ist möglich. Sei o.B.d.A. $B > C_2$, d.h. alle Ereignisse von B treten nach C_2 auf (siehe Abb. A.1).

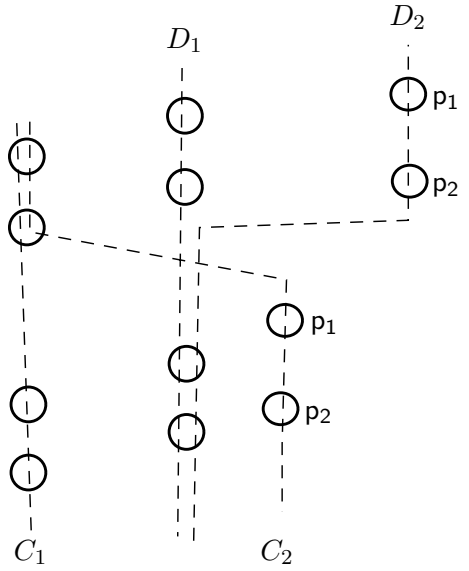


Abb. A.1: Der Fall $C_1 \nmid B$ und $D_1 \mid A$

Wir betrachten nun den Schnitt $C = \sup(C_2, D_2)$. D_2 enthält zu jeder Stelle aus dem Nachbereich der ersetzten Transition t eine Bedingung, die mit dieser Stelle beschriftet ist (wegen Bedingung (ii) aus Satz 2.2). In Abb. A.1 haben wir angenommen, daß $t^\bullet = \{p_1, p_2\}$ ist. Da alle Ereignisse von B nach C_2 auftreten, sind diese Bedingungen in C. Da es mindestens ein Ereignis von A gibt, das nach D_2 auftritt, gibt es auch mindestens eine Bedingung in C_2, die mit einer Stelle aus dem Nachbereich von t beschriftet ist und die außerdem in C ist. Es gibt also zwei Stellen in C, die mit der gleichen Stelle aus dem Nachbereich von t beschriftet sind. Dies ist ein Widerspruch dazu, daß $\Sigma\big[t \to N_E\big]$ sicher ist.

Wir betrachten nun den schwierigen Fall, daß sowohl $C_1 \mid B$ (also auch $C_2 \mid B$), als auch $D_1 \mid A$ (also auch $D_2 \mid A$) gilt. Um Behauptung 2 für diesen Fall zu beweisen, stellen wir einige Zwischenbehauptungen auf.

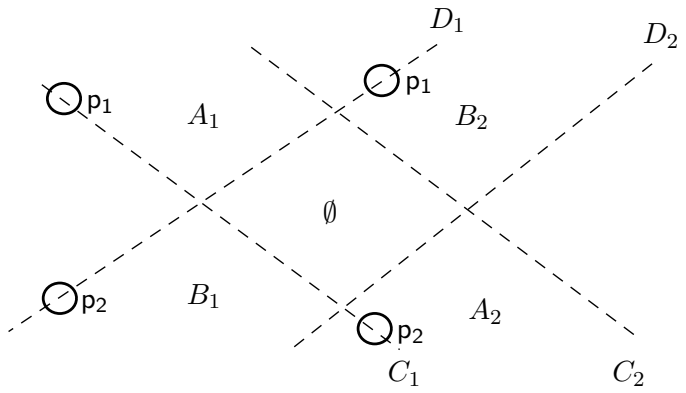


Abb. A.2: Der Fall $C_1 \mid B$ und $D_1 \mid A$

Die Schnitte C_1 und C_2 teilen die Äquivalenzklasse B in zwei disjunkte Mengen B_1 und B_2, so daß $B_1 < C_1, C_2$ und $B_2 > C_1, C_2$ und $B_1 \cup B_2 = B$. Analog teilen D_1 und D_2 die Äquivalenzklasse A in zwei disjunkte Mengen A_1 und A_2, so daß $A_1 < D_1, D_2$ und $A_2 > D_1, D_2$ und $A_1 \cup A_2 = A$. Wir haben diesen Fall in Abb. A.2 skizziert. In dem Viereck, das in der Mitte der Abbildung durch die sich kreuzenden Schnitte aufgespannt wird, liegt überhaupt kein Ereignis, denn es wäre dann in beiden Äquivalenzklassen. Die Äquivalenzklassen einer Partition sind aber disjunkt.

Mit Hilfe der folgenden Zwischenbehauptungen werden wir zeigen, daß $\tilde{A} = A_1 \cup B_1$ und $\tilde{B} = A_2 \cup B_2$ die Bedingungen von Behauptung 2 erfüllen.

Da das System sicher ist, sind für jede Stelle p aus $\Sigma\big[t \to N_E\big]$ alle Bedingungen, die mit dieser Stelle beschriftet sind, linear geordnet in ρ'. Da beide Äquivalenzklassen einen Ablauf des Ersetzungsnetzes induzieren, gibt es in C_1 und D_1 zu jeder Stelle aus dem Vorbereich von t eine Bedingung, die mit dieser Stelle beschriftet ist. Wir zeigen nun, daß die Bedingungen aus den Vorbereich von t "über Kreuz" geordnet sind, d.h. wenn p aus dem Vorbereich von t ist und die mit p beschriftete Bedingung aus C_1 hat ein Ereignis aus A_1 im Nachbereich, dann hat die die mit p beschriftete Bedingung aus D_1 ein Ereignis aus B_2 im Nachbereich u.s.w. (siehe Abb. A.2)

Behauptung 3: Seien $c \in C_1$ und $d \in D_1$ mit $r'(c) = r'(d) = p \in {}^\bullet t$, dann gilt

$$c^\bullet \in A_1 \qquad \Longleftrightarrow \qquad d^\bullet \in B_2.$$

Beweis: Da alle mit mit p beschriften Ereignisse linear geordnet sind, gilt $c < d$ oder $d < c$.
($\Longrightarrow$) Angenommen, $c^\bullet \in A_1$. Da kein Ereignis von B vor einem Ereignis aus A_1 liegt, muß $c < d$ gelten. Dann gilt aber $d > C_1$, also ist $d^\bullet \in B_2$.
($\Longleftarrow$) Angenommen, $d^\bullet \in B_2$. Da kein Ereignis von A nach einem Ereignis aus B_2 liegt, muß $c < d$ gelten. Dann gilt aber $c < D_1$, also ist $c^\bullet \in A_1$. Damit ist Behauptung 3 bewiesen.

Das gleiche gilt für die Stellen aus dem Nachbereich von t:

Behauptung 4: Seien $c \in C_2$ und $d \in D_2$ mit $r'(c) = r'(d) = p \in t^\bullet$, dann gilt

$${}^\bullet c \in A_1 \qquad \Longleftrightarrow \qquad {}^\bullet d \in B_2.$$

Der Beweis dieser Behauptung ist analog zum Beweis von Behauptung 3.

Behauptung 5: Es gilt $A_1 \operatorname{co} A_2$ und $B_1 \operatorname{co} B_2$.

Beweis: Angenommen, es gibt ein $e_1 \in A_1$ und ein $e_2 \in A_2$, so daß nicht $e_1 \operatorname{co} e_2$ gilt. Wegen der Definition von A_1 und A_2 muß dann $e_1 < e_2$ gelten. Da D_1 und D_2 gerade A in A_1 und A_2 teilen und weil kein Ereignis von A zwischen D_1 und D_2 liegt, gibt es eine Bedingung d_1 mit $e_1 < d_1 < e_2$ und $d_1 \in D_1$ und $d_1 \in D_2$ (siehe Abb. A.3). Wegen Behauptung 3 gilt $r'(C_1 \cap {}^\bullet A_1) = r'(D_1 \cap {}^\bullet B_2)$. Sei $C = D_1 \cap {}^\bullet B_2$. C ist eine co-Menge.

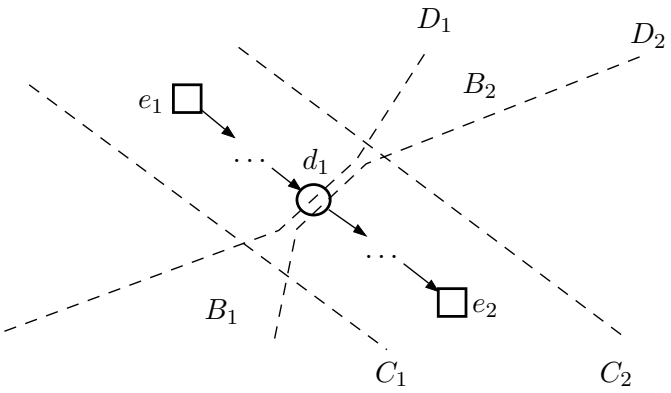


Abb. A.3: Der Fall $C_1 \mid B$ und $D_1 \mid A$

Wir betrachten nun einen anderen Ablauf ρ'' von $\Sigma\big[t \to N_E\big]$, der bis zum Schnitt D_1 identisch mit ρ' ist. An die co-Menge $C \subseteq D_1$ hängen wir aber nun nicht die Ereignisse von B_2 an, sondern noch einmal Ereignisse, die genau wie in A_1 beschriftet sind. Irgendwann erreichen wir eine Bedingung d_2 mit $r'(d_1) = r'(d_2)$. Die Bedingung d_1 aus D_1 wurde bis zur Erzeugung von d_2 noch nicht verbraucht, dies ist jedoch ein Widerspruch dazu, daß $\Sigma\big[t \to N_E\big]$ sicher ist. Damit ist Behauptung 5 bewiesen.

Aus Behauptung 5 folgt, daß jede der beiden Äquivalenzklassen A und B zwei zueinander nebenläufige Teilabläufe des Ersetzungsnetzes induziert. Aus den Behauptungen 3 und 4 folgt, daß wir diese Teilabläufe paarweise vertauschen können, da Anfang und Ende des von A_1 induzierten Ablaufs mit Anfang und Ende des von B_2 induzierten Ablaufs übereinstimmen u.s.w. Damit erfüllen $\tilde{A} = A_1 \cup B_1$ und $\tilde{B} = A_2 \cup B_2$ die Bedingungen von Behauptung 2. Damit ist Behauptung 2 bewiesen.

Wir können also je zwei Äquivalenzklassen so umordnen, daß es begrenzende Schnitte gibt, die die jeweils andere Äquivalenzklasse nicht schneiden. Um eine Partition zu erhalten, die die Bedingungen (i)(a) und (ii) aus Satz 2.2 und außerdem (i)(b) erfüllt, ordnen wir nun die Äquivalenzklassen nach und nach um. Um "mit der kleinsten" zu beginnen, definieren wir zuerst eine lineare Ordnung auf den Äquivalenzklassen der Partition. Sei p eine beliebige (aber von nun an feste) Stelle aus dem Vorbereich von t. Wie bereits erwähnt hat jede Äquivalenzklasse eine mit p beschriftete Bedingung im Vorbereich und alle mit p beschrifteten Bedingungen sind linear geordnet. Wir erhalten also eine lineare Ordnung der Äquivalenzklassen, indem wir definieren $A \prec B$ genau dann, wenn für alle $a \in {}^\bullet A$ und $b \in {}^\bullet B$ gilt: Falls $r'(a) = r'(b) = p$, dann ist $a < b$.

Da das System sicher ist, wird jede Äquivalenzklasse nur von endlich vielen Schnitten

anderer Äquivalenzklassen geteilt. Wir beginnen also mit der kleinsten Äquivalenzklasse A und ordnen A paarweise mit jeder anderen Klasse, deren Schnitte A teilen um, wie in Behauptung 2 beschrieben. Wir erhalten schließlich eine Partition, die alle Bedingungen von Satz 2.2 erfüllt.

A.2 Graphen und Bäume

In dieser Arbeit benutzen wir einen Graphen meist als Abstraktion eines Kommunikationsnetzwerks. Wir betrachten deshalb nur endliche Graphen und halten dies (abweichend von klassischen Definitionen) bereits in der Definition fest.

Definition A.2

Ein *Graph* (A, N) besteht aus einer nichtleeren endlichen Menge A von Agenten und einer Menge $N \subseteq A \times A$ von Kanten. Wir nennen $\overline{N} = \{(x, y) \mid (y, x) \in N\}$ die Menge der invertierten Kanten des Graphen.

- Wir nennen einen Graphen *ungerichtet* genau dann, wenn N irreflexiv ist und $N = \overline{N}$ gilt. Dabei wird eine ungerichtete Kante zwischen den Knoten x und y durch die beiden geordneten Paare (x, y) und (y, x) repräsentiert.
- Ein *Pfad* in (A, N) ist eine Folge $(x_0, \ldots, x_n)$ von Elementen aus A, so daß für alle $0 \leq i < n$ gilt: $(x_i, x_{i+1}) \in N$.
- Ein *Weg* ist ein Pfad, in dem $x_i \neq x_j$ für alle $i \neq j$ gilt.
- Ein *Kreis* ist ein Weg $(x_0, \ldots, x_n)$, für den $(x_n, x_0) \in N$ gilt.
- Ein ungerichteter Graph (A, N) ist *azyklisch* genau dann, wenn es in (A, N) keinen Kreis $(x_0, \ldots, x_n)$ mit $n \geq 2$ gibt.
- Ein Graph (A, N) ist *zusammenhängend* genau dann, wenn $(N \cup \overline{N})^* = A \times A$ gilt. (Hier bezeichnet * die reflexive transitive Hülle)
- Ein Graph (A, N) ist *stark zusammenhängend* genau dann, wenn $N^* = A \times A$ gilt.
- Ein *Baum* ist ein ungerichteter Graph (A, N), der azyklisch und zusammenhängend ist.
- Ein Baum (A, N) ist ein *Stern*, wenn es einen Knoten $a \in A$ gibt, so daß $N = (A \setminus \{a\}) \times \{a\} \cup \{a\} \times (A \setminus \{a\})$.
- Ein *Wurzelbaum mit der Wurzel* r ist ein zusammenhängender Graph (A, N), in dem für alle Knoten x gilt: $(r, x) \notin N$ und für alle $x \neq r$ gibt es genau einen Knoten y, so daß $(x, y) \in N$ gilt.
- Ein Graph (A', N') heißt *Teilgraph* von (A, N), wenn $A' \subseteq A$ und $N' \subseteq N$ gilt.
- Ein Graph (A', N') heißt *Teilbaum* von (A, N), wenn (A', N') ein Teilgraph von (A, N) und ein Baum ist. Ein Teilbaum heißt spannender Baum, wenn $A' = A$ ist.

- Sei $g : N \longrightarrow \mathbb{R}$ eine Funktion. Dann heißt (A, N, w) *gewichteter Graph* mit der Gewichtungsfunktion w. Ein ungerichteter, gewichteter Graph (A, N, w) heißt *eindeutig gewichtet*, wenn verschiedene Kanten verschiedene Gewichte haben, also wenn für alle $(x, y) \in N$ gilt:

$$w(x_1, y_1) = w(x_2, y_2) \qquad \text{gdw.} \qquad \{x_1, y_1\} = \{x_2, y_2\}.$$

- Ein spannender Baum (A, B) des Graphen (A, N) ist ein *minimaler spannender Baum* des gewichteten Graphen (A, N, w), wenn für jeden anderen spannenden Baum (A, B') des Graphen gilt: $\sum_{(x,y)\in B} w(x, y) \leq \sum_{(x,y)\in B'} w(x, y)$ $\star$

Bemerkungen

- Wir nennen ein Kommunikationsnetzwerk zusammenhängend, eindeutig gewichtet etc. wenn der zugrunde liegende ungerichtete Graph zusammenhängend, eindeutig gewichtet etc. ist.
- Jeder ungerichtete Graph ist genau dann stark zusammenhängend, wenn er zusammenhängend ist.
- Der Knoten y ist genau dann von x erreichbar, wenn $(x, y) \in N^*$.
- Wenn (A, N) ein Wurzelbaum ist, dann ist $(A, N \cup \overline{N})$ ein Baum.
- Für jeden Wurzelbaum (A, N) gilt: $|A| = |N| + 1$, denn nach Definition geht von jedem Knoten außer der Wurzel genau eine Kante aus. Für jeden Baum gilt $2|A| = |N| + 2$.
- Jeder Wurzelbaum ist zusammenhängend.

A.3 Stotterfreie Version einer Folge

Sei $\sigma = (x_0, x_1, \dots)$ eine Folge. Die stotterfreie Version von σ definieren wir folgendermaßen: Wir definieren zuerst induktiv eine Folge geordneter Paare $\sigma' = ((y_0, z_0), (y_1, z_1), \dots)$ durch:

(i) $(y_0, z_0) = (x_0, 0)$.

(ii) Sei (y_i, z_i) bereits definiert. Wenn es eine natürliche Zahl k gibt mit

$$k = \min\{m > z_i \mid y_i \neq x_m\}$$

dann definieren wir $(y_{i+1}, z_{i+1}) = (x_k, k)$. Ansonsten ist (y_{i+1}, z_{i+1}) nicht definiert und die Folge σ' hat genau $i + 1$ Elemente.

Die stotterfreie Version von σ ist dann $\natural\sigma = (y_0, y_1, \dots)$ die Folge der ersten Elemente der geordneten Paare aus σ'.

Literaturverzeichnis

[AL91] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82:253–284, 1991. previously: SRC Research Report 27, April 1988.

[AS85] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21:181–185, October 1985.

[AW98] Hagit Attiya and Jennifer Welch. *Distributed Computing*. McGraw-Hill, 1998.

[Bac88] Ralph J. R. Back. A calculus of refinement for program derivations. *Acta Informatica*, 15:233–249, 1988.

[Bar96] Valmir C. Barbosa. *An Introduction to Distributed Algorithms*. The MIT Press, Cambridge, Massachusetts, 1996.

[BGV90] Wilfried Brauer, Robert Gold, and Walter Vogler. A survey of behaviour and equivalence preserving refinement of Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, LNCS 483, pages 1–46. Springer-Verlag, 1990.

[BKV99] Thomas Baar, Ekkart Kindler, and Hagen Völzer. Verifying intuition - ilf checks dawn proofs. In W. Reisig and G. Rozenberg, editors, *Proceedings of the International Conference on Application and Theory of Petri Nets (ICAPTN 99)*, LNCS. Springer, June 1999.

[BM88] R. S. Boyer and J. Moore. *A Computational Logic Handbook*. Academic Press, Boston, 1988.

[BS96] R. J. R. Back and K. Sere. Superposition refinement of reactive systems. *Formal Aspects of Computing*, 8:324–346, 1996.

[BT97] Eike Best and Thomas Thielke. Refinement of coloured petri nets. In B.S.Chlebus and L.Czaja, editors, *Fundamentals of Computation Theory. Proceedings of 11th International Symposium, FCT'97*, LNCS 1279, pages 105–116. Springer-Verlag, 1997.

[CG88] Ching-Tsun Chou and Eli Gafni. Understanding and verifying distributed algorithms using stratified decomposition (extended abstract). In *Proceedings of the 7th Annual Symposium on Principles of Distributed Computing*, pages 44–65. ACM, 1988.

[CM88] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[CMP87] Luca Castellano, Giorgio De Michelis, and Lucia Pomello. Concurrency vs interleaving: an instructive example. *EATCS Bulletin*, (31):12–15, February 1987.

[Des97] Jörg Desel. How distributed algorithms play the token game. In Freska e.a., editor, *Foundations of Computer Science: Potential–Theory–Cognition*, volume 1337 of *LNCS*, pages 297–306. Springer Verlag, 1997.

[dRE98] Willem-Paul de Roever and Kai Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison*, volume 47 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[EF82] T. Elrad and N. Francez. Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2(3):155–173, 1982.

[FM82] A. Finkel and G. Memmi. Fifo nets: A new model of parallel computation. In *Lecture Notes in Computer Science: 6th GI-Conference on Theor. Comp. Science, Dortmund*, volume 145, pages 111–121. Springer-Verlag, 1982. NewsletterInfo: 15.

[GHS83] R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and Systems*, 5(1):66–77, January 1983.

[GKS92] Rob Gerth, Ruurd Kuiper, and John Segers. Interface refinement in reactive systems (extended abstract). In W. R. Cleaveland, editor, *CONCUR '92*, volume 630 of *LNCS*. Springer-Verlag, 1992.

[Gri77] David Gries. An exercise in proving parallel programs correct. *Communication of the ACM*, 20(12):921–930, 1977.

[Gri93] E. Pascal Gribomont. Concurrency without toil : a systematic method for parallel program design. *Science of Computer Programming*, 21:1–56, 1993.

[Gri96] E. Pascal Gribomont. Atomicity refinement and trace reduction theorems. In *Proc. 8th Conf. on Computer Aided Verification*, volume 1102 of *Lecture Notes in Computer Science*, pages 311–322, Rutgers University, New Jersey, USA, August 1996. Springer-Verlag.

[Hes99a] W. H. Hesselink. The incremental design of a distributed spanning tree algorithm. *Formal Aspects of Computing*, 11(E):(electronic archive), 1999.

[Hes99b] W. H. Hesselink. The verified incremental design of a distributed spanning tree algorithm: extended abstract. *Formal Aspects of Computing*, 11:45–55, 1999.

[HHS87] C.A.R. Hoare, Jifeng He, and Jeff W. Sanders. Prespecification in data refinement. *Information Processing Letters*, 25:71–76, 1987.

[Jan94] Wil Janssen. *Layered Design of Parallel Systems*. Phd thesis, University Twente, Enschede, 1994.

[Jen92] Kurt Jensen. *Coloured Petri Nets*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1992.

[JMZ91] W. Janssen, M.Poel, and J. Zwiers. Action systems and action refinement in the development of parallel systems. In J. C. M. Baeten and J. F. Groote, editors, *CONCUR '91*, volume 527 of *LNCS*, pages 298–316, 1991.

[Jon91] Bengt Jonsson. Simulations between specifications of distributed systems. In J. C. M. Baeten and J. F. Groote, editors, *Proceedings of CONCUR '91*, volume 527 of *LNCS*, 1991.

[JZ92] Wil Janssen and Job Zwiers. From sequential layers to distributed processes, deriving a distributed minimum weight spanning tree algorithm (extended abstract). In *Proceedings of the 11th Annual Symposium on Principles of Distributed Computing*, pages 215–227. ACM, 1992.

[Kin97] Ekkart Kindler. A compositional partial order semantics for Petri net components. In Pierre Azema and Gianfranco Balbo, editors, *18th Conf. of Application and Theory of Petri Nets*, volume 1248 of *LNCS*. Springer, 1997.

[KP90] Shmuel Katz and Doron Peled. Interleaving set temporal logic. *Theoretical Computer Science*, 75(3):263–287, 1990.

[KP92] Shmuel Katz and Doron Peled. Verification of distributed programs using representative interleaving sequences. *Distributed Computing*, 6:107–120, 1992.

[KP99] Ekkart Kindler and Sibylle Peuker. Integrating distributed algorithms into distributed systems. *Fundamenta Informaticae*, 37:291–309, 1999.

[KR96] Ekkart Kindler and Wolfgang Reisig. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter*, 51:16–31, December 1996.

[KV98] E. Kindler and H. Voelzer. Flexibility in algebraic nets. In *Proceedings of ICATPN '98*, volume 1420 of *LNCS*, pages 345–364, 1998. Erscheint in *Theoretical Computer Science*.

[KW99] Ekkart Kindler and Michael Weber. The Petri Net Kernel: An infrastructure for building petri net tools. In *Petri Nets '99. 20th International Conference on Application and Theory of Petri Nets. Petri Net Tool Presentations*, Williamsburg, USA, June 1999. `http://www.informatik.hu-berlin.de/top/pnk/`.

[LL90] Leslie Lamport and Nancy Lynch. Distributed computing: Models and methods. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, chapter 18, pages 1158–1199. Elsevier Science Publishers B.V., 1990.

[LT87] Nancy R. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, 1987.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Mat89] Friedemann Mattern. *Verteilte Basisalgorithmen*. Number 226 in Informatik-Fachberichte. Springer-Verlag, 1989.

[ME92] Carroll Morgan and Trevor Vickers (Eds.). *On the Refinement Calculus*. FACIT. Springer-Verlag, 1992.

[Peu99] Sibylle Peuker. Phased decomposition of distributed algorithms. Informatik-Bericht 120, Humboldt-Universität zu Berlin, 1999.

[Ray88] Michel Raynal. *Distributed Algorithms and Protocols*. Wiley series in computing. Wiley, 1988.

[Rei85] Wolfgang Reisig. *Petri Nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[Rei91] W. Reisig. Petri Nets and Algebraic Specifications. *Theoretical Computer Science*, 80:1–34, May 1991.

[Rei98] Wolfgang Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer, 1998.

[SdR87] F. A. Stomp and W. P. de Roever. A correctness proof of a distributed minimum-weight spanning tree algorithm (extended abstract). In *Proceedings of the 7th International Conference on Distributed Computing Systems*, pages 440–447. ACM, 1987.

[SdR94] F. A. Stomp and W. P. de Roever. Principles for sequential reasoning about distributed algorithms. *Formal Aspects of Computing*, 6E:1–70, 1994.

[Seg83] A. Segall. Distributed network protocols. *IEEE Transactions on Information Theory*, IT-29(1), 1983.

[Sie96] Michael Siegel. *Phased Design and Verification of Stabilizing Systems*. Dissertation, tech.rep. 9705, Christian-Albrechts-Universität Kiel, July 1996.

[Sta83] Peter H. Starke. Monogenous FIFO-nets and petri-nets are equivalent. *EATCS Bull.*, (21):68–77, 1983.

[Sto89] Frank A. Stomp. *Design and Verification of Distributed Network Algorithms: Foundations and Applications*. PhD thesis, Technische Universiteit Eindhoven, 1989.

[Tel91] Gerard Tel. *Topics in Distributed Algorithms.* Cambridge University Press, 1991.

[Tel94] Gerard Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, Cambridge, 1994.

[Tix96] Petra Tix. One FIFO place realizes zero-testing and turing-machines. *Petri Net Newsletter*, (51):3–15, December 1996.

[Val79] R. Valette. Analysis of Petri Nets by stepwise refinement. *Journal of Computer and System Sciences*, 18:35–46, 1979.

[vdA98] W. M. P. van der Aalst. The application of Petri nets to workflow management. *The Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[vdMM98] Ron van der Meyden and Yoram Moses. Top-down considerations on distributed computing. In Shay Kutten, editor, *Distributed Computing. 12th International Symposium, DISC'98*, volume 1499 of *LNCS*, pages 16–19. Springer-Verlag, September 1998.

[vGG89] R. J. van Glabbeek and U. Goltz. Partial order semantics for refinement of actions – neither necessary nor always sufficient but appropriate when used with care. *Bulletin of the European Association for Theoretical Computer Science*, 38:154–163, June 1989.

[vGG90] R. J. van Glabbeek and U. Goltz. Refinement of actions in causality based models. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness,* Mook, The Netherlands, May/June 1989, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer-Verlag, 1990.

[Vog87] Walter Vogler. Behaviour preserving refinements of Petri nets. In G. Tinhofer and G. Schmidt, editors, *Graph-Theoretic Concepts in Computer Science*, volume 246 of *LNCS*, pages 82–93. Springer Verlag, 1987.

[Vog93] Walter Vogler. Bisimulation and action refinement. *Theoretical Computer Science*, 114:173–200, 1993.

[Wal95] Rolf Walter. *Petrinetzmodelle verteilter Algorithmen: Beweistechnik und Intuition.* Phd thesis, Humboldt-Universität zu Berlin, 1995.

[Wel88] Jennifer L. Welch. *Topics in Distributed Computing.* Phd thesis, MIT/LCS/TM-361, 1988.

[Wir71] Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.

[WLL88] Jennifer Lundelius Welch, Leslie Lamport, and Nancy Lynch. A lattice-structured proof technique applied to a minimum spanning tree algorithm (extended abstract). In *Proceedings of the 7th Annual Symposium on Principles of Distributed Computing*, pages 28–43. ACM, 1988.

[WWV+97] M. Weber, R. Walter, H. Völzer, T. Vesper, W. Reisig, S. Peuker, E. Kindler, J. Freiheit, and J. Desel. DAWN: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. Informatik-Bericht 88, Humboldt-Universität zu Berlin, December 1997.

[Zwi98] Job Zwiers. Compositional transformational design for concurrent systems. In Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli, editors, *Compositionality: The Significant Difference, Proc. of COMPOS '97*, volume 1536 of *LNCS*, pages 609–631. Springer-Verlag, 1998.

Index